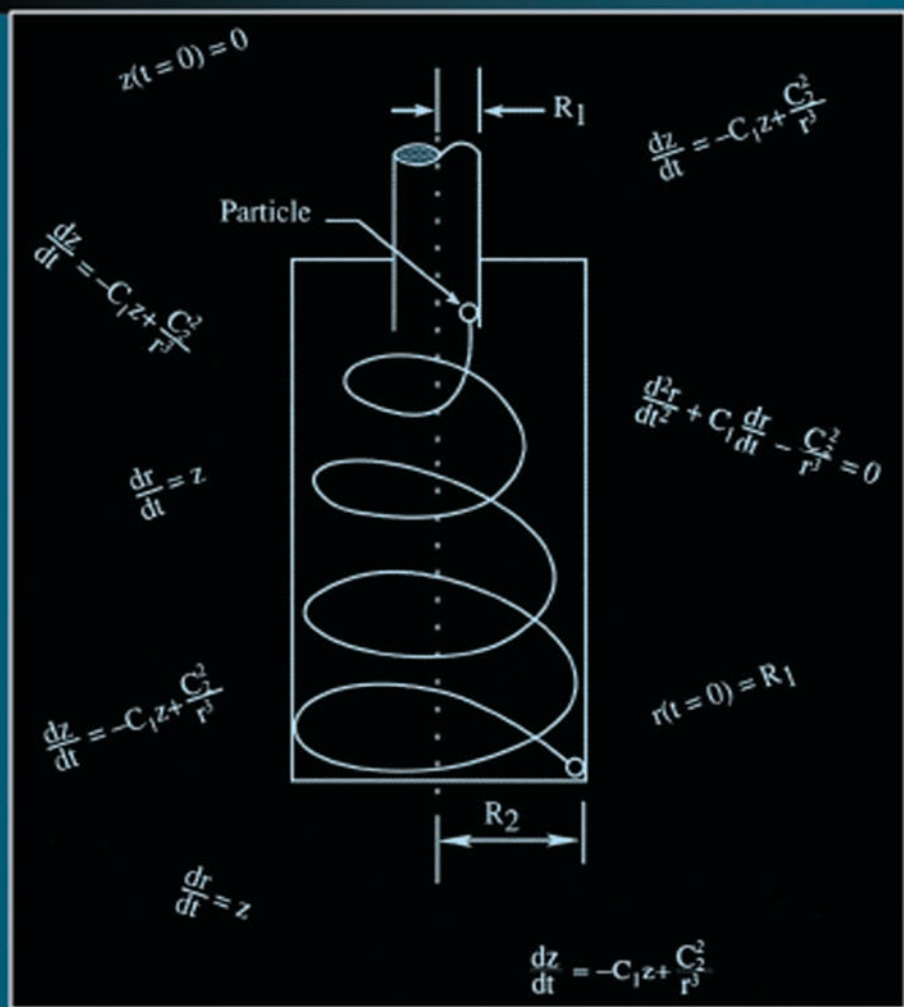


Applied Numerical Methods for Food and Agricultural Engineers



Prabir K. Chandra and R. Paul Singh

VISIT...

LANZAROTE
Caliente.COM

VISIT...

LANZAROTE
Caliente.COM

Applied Numerical Methods for Food and Agricultural Engineers

Prabir K. Chandra, Ph.D.

Department of Food Science
Agricultural Institute of Lavras (ESAL)
Lavras, Brazil

and

R. Paul Singh, Ph.D.

Department of Biological and Agricultural Engineering
University of California, Davis
Davis, California



CRC Press

Boca Raton Ann Arbor London Tokyo

Library of Congress Cataloging-in-Publication Data

Chandra, Prabir K.

Applied numerical methods for food and agricultural engineers /
Prabir K. Chandra and R. Paul Singh.

p. cm.

ISBN 0-8493-2454-8

I. Numerical analysis. I. Singh, R. Paul. II. Title.
QA300.C4415 1994

519.4--dc20

for Library of Congress

94-29202

CIP

DISCLAIMER OF WARRANTY AND LIMITS OF LIABILITY: The authors of this book have used their best efforts in preparing this material. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. Neither the author nor the publisher make warranties of any, express or implied, with regard to these programs, or the documentation contained in this book, including without limitation warranties of merchantability or fitness for a particular purpose. No liability is accepted in any event for any damages, including incidental or consequential damages, lost profits, costs of lost data or program material, or otherwise in connection with or arising out of the furnishing, performance, or use of the programs in this book.

This book contains information obtained from authentic and highly regarded sources. Reprinted material is quoted with permission, and sources are indicated. A wide variety of references are listed. Reasonable efforts have been made to publish reliable data and information, but the author and the publisher cannot assume responsibility for the validity of all materials or for the consequences of their use.

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without prior permission in writing from the publisher.

CRC Press, Inc.'s consent does not extend to copying for general distribution, for promotion, for creating new works, or for resale. Specific permission must be obtained in writing from CRC Press for such copying.

Direct all inquiries to CRC Press, Inc., 2000 Corporate Blvd., N.W., Boca Raton, Florida 33431.

© 1995 by CRC Press, Inc.

No claim to original U.S. Government works

International Standard Book Number 0-8493-2454-8

Library of Congress Card Number 94-29202

Printed in the United States of America 1 2 3 4 5 6 7 8 9 0

Printed on acid-free paper

PREFACE

In the study of food and agricultural engineering it is necessary to be familiar with the tools that are required to develop the simulation of physical and/or chemical processes in order for implementation by a computer. The number crunching machine can then produce the desired information to help better understand the processes. The essence of mathematical modeling of physical processes lies in the fact that it offers the advantage of analyzing effects of experimental variables among themselves. Thus an elaborate, expensive, and time-consuming experimentation is not necessary every time.

Most of the natural and mechanical phenomena are complex. They cannot be expressed analytically; they do not have simple and straightforward exact solutions. One branch of mathematics that deals with the art of finding approximate solutions of such phenomena is called numerical mathematics — numerical because it transforms the complicated problems into simple and discrete ones, therefore facilitating the stepwise solution by powerful digital computers.

In order to conduct required computational and related procedures on a computer, several computer languages have been developed. FORTRAN was chosen for the purposes of this book, the obvious reason being that there does not exist a better language for mathematical computations

This book has been divided mainly into five parts. The first part orients its readers towards different unit systems. It incorporates the basics of its conversion from one unit to another. The second part gives a concise treatment of the FORTRAN language. The authors believe that this provides a useful resource; one need not consult another volume to learn the language utilized in this text. Though the treatment has been concise, every effort has been made to cover the salient features of FORTRAN. Sufficient examples are provided to ease the task of learning and understanding. Emphasis has been placed on a structured and disciplined programming style without referring to a particular compiler version.

The third part deals with the numerical methods, starting with finite difference and continuing with root finding, integration, solution of simultaneous linear algebraic equation with matrix concept, least squares approximations, solution of ordinary differential equations of any number and order, solution of partial differential equations in one and two dimensions, and finite element methods. In conclusion it includes a limited discussion on errors. All methods have been discussed in detail and complete program listings pertinent to each chapter (with the exception of the finite element methods) have been provided in the Appendix.

Dimensional analysis is, frequently, a very convenient tool of mathematical modeling. This subject, in spite of its widespread use and simplicity, has not received proper attention in food or agricultural engineering curricula. The objective of this book has necessitated inclusion of this uncommon topic in the fourth part.

The fifth, and final, part on applications is designed to present detailed solutions of some very common problems encountered in the field of food and agricultural engineering. In all of the cases, the methods learned in the

book, and the subroutines developed, have been used to solve these problems and complete program listings are provided. This part includes (i) air psychrometrics, (ii) finned tube heat exchanger, (iii) heat transfer in sand puffing, (iv) mechanical separation by cyclone, and (v) continuous drying involving the diffusion and thin-layer concepts. The computer programs listed in this book are available on a disk for a nominal charge from the first author.

The present work is intended to assist its readers acquire the basics needed in the field of mathematical modeling and much more. Though the title suggests only two disciplines, it will have much wider applications in almost all the branches of engineering.

Dr. Chandra is indebted to CNPq (National Research Council of Brazil) for awarding him the prestigious Research Scholarship (Bolsa de Pesquisa) during the work of writing this book. He is also grateful to his colleagues and students in the Food Science Department (DCA) at ESAL for their constant encouragement and for providing a conducive environment.

Prabir K. Chandra
R. Paul Singh

DEDICATION

*To our parents,
who gave us the opportunities to develop a love of learning
and
to our wonderful families,
Krishna, Neelav, Amick, Evan, Anila, and Raj*

THE AUTHORS

Prabir K. Chandra, Ph.D., is a Visiting Professor in the Food Science Department of the Agricultural Institute of Lavras (ESAL) in Brazil.

Dr. Chandra earned his bachelors and masters degrees in Agricultural Engineering from the Indian Institute of Technology (IIT) at Kharagpur in 1970 and 1972, respectively. In 1983, he received his Ph.D. degree from the University of California, Davis, in Food Engineering. During his studies at Davis, he maintained a status of a Distinguished Scholar. After earning his Ph.D., he conducted his postdoctoral work at Clemson University for more than two years. Subsequently, he returned to India and was appointed at the level of an Associate Professor at the IIT, Kharagpur, where he taught food engineering courses at both the undergraduate and graduate levels. He has had vast working experiences in India, Japan, and the United States. Dr. Chandra is a member of the American Society of Agricultural Engineers (ASAE).

Dr. Chandra has published over 40 research papers. One of them, on the mathematical modeling of an updraft gasifier-combustor, received the First Author Paper Award from the ASAE in 1988. The idea of writing the present book surfaced when he first taught a portion of numerical methods at Clemson University. The real inspiration came when, later at IIT, he regularly taught computational methods at the graduate level. In Brazil, he introduced this topic into the graduate curriculum. His current research interests include mathematical modeling of physical processes and unit operations.

R. Paul Singh, Ph.D., is a Professor of Food Engineering, Department of Biological and Agricultural Engineering, Department of Food Science and Technology, University of California, Davis.

Dr. Singh graduated in 1970 from Punjab Agricultural University, Ludhiana, India, with a degree in Agricultural Engineering. He obtained an M.S. degree from the University of Wisconsin, Madison, and a Ph.D. degree from Michigan State University in 1974. Following a year of teaching at Michigan State University, he moved to the University of California, Davis, in 1975 as an Assistant Professor of Food Engineering. He was promoted to Associate Professor in 1979 and, again, to Professor in 1985.

Dr. Singh is a member of the Institute of Food Technologists, American Society of Agricultural Engineers, and Sigma Xi. He received the A.W. Farrall Young Educator Award, American Society of Agricultural Engineers in 1986. He was a NATO Senior Guest Lecturer in Portugal in 1987 and 1993, and received the IFT International Award, Institute of Food Technologists, 1988, and the Distinguished Alumnus Award from Punjab Agricultural University in 1989.

Dr. Singh has authored and co-authored seven books and over 150 technical papers. He is a co-editor of the Journal of Food Process Engineering. His current research interests are in studying transport phenomena in foods as influenced by structural changes during processing.

CONTENTS

PART I. UNITS AND DIMENSIONS	1
Concepts	2
Symbols	9
References	10
 PART II. A CONCISE TREATISE ON FORTRAN	 11
Number systems	12
Structured programming	13
Variables and constants	14
Operation symbols	15
Arithmetic assignments	16
Coding FORTRAN statements	18
Programming in FORTRAN	18
STOP and END statements	19
List-directed input and output	19
Processing more than one datum	22
A logical IF statement	23
Relational operators	23
Logical operators	23
Block IF statement	27
Simple DO-loop	28
Flow symbols/chart/diagram	28
Arrays — subscripted variables	32
DIMENSION statement	32
PARAMETER statement	33
External input and output files	36
Implied DO-loops in READ and WRITE	38
Checking for end of data by READ statement	38
FORMAT-directed input and CHARACTER statement	40
FORMAT-directed output	42
Repetition of FORMAT specifications	45
Branched block IF	47
Intrinsic/library functions	49

REAL, INTEGER, and DOUBLE PRECISION statements	49
DO WHILE and EXIT statements	49
Arithmetic IF and computed GOTO statements.....	53
DATA statement	53
EQUIVALENCE statement	54
Subprograms — FUNCTIONs and SUBROUTINEs	55
Differences between a Program and a Subprogram	56
Layout of a program	59
EXTERNAL statement	59
COMMON statement.....	60
Arithmetic statement or line function.....	62
BLOCK DATA subprogram.....	65
ENTRY statement.....	66
Variable Input/Output file name	66
REWIND statement	66
PAUSE statement	67
Metacommand \$INCLUDE.....	67
References	76

PART III. NUMERICAL METHODS 77

Chapter 3.1 Numerical Differentiation	77
Taylor Series expansion.....	78
Finite difference formulas for differentiation	79
Richardson extrapolation.....	82
Anatomy of a numerical differentiation.....	86
References	88
Chapter 3.2 Real Roots of Algebraic Equations	89
Iterative method.....	90
Newton-Raphson method	91
A search for the region where the zeros lie.....	93
Modified Newton method.....	93
Secant method	94
Convergence criteria.....	95
Bisection method	95
False Position method.....	97
False Position method (improved).....	98

Development of a program to compute a root	98
Development of a general program to compute all roots	100
Discussion of results	101
References	103
 Chapter 3.3 Numerical Integration	 105
Trapezoidal rule.....	106
Simpson's rule.....	109
Romberg method	111
Gauss-Legendre quadrature	116
Approximations to singularities	119
Improper integral of the first kind.....	119
Improper integral of the second kind	120
Development of a computer program	122
References	123
 Chapter 3.4 Solution of Linear Algebraic Equations	 125
Arithmetic manipulations with matrices	127
Addition of matrices.....	127
Subtraction between matrices	128
Multiplication of matrices.....	128
Inverse of a matrix	130
Transpose of a matrix	130
Orthogonality of a matrix	131
Determinant of a matrix	131
Rank of a matrix	133
Solution of a set of linear algebraic equations	133
Gauss elimination method	135
Back substitution	137
Pivotal condensation.....	139
Gauss-Jordan elimination	141
Matrix inversion method.....	143
Gauss-Seidel iteration.....	146
Criteria for convergence	149
Relaxation in Gauss-Seidel iterative method	149
Gauss elimination method in solving banded matrix	151
Development of computer programs	153
References	155

Chapter 3.5 Least Squares Approximation	157
Linear regression	158
Multilinear regression.....	160
Trigonometric regression.....	162
Page's equation	162
Asymptotic regression	163
Logistic regression.....	164
Polynomial regression	165
Multiple correlation coefficient	167
Standard error.....	167
References	168
 Chapter 3.6 Solution of Initial and Boundary Value Problems.....	 169
Initial value problems	171
Euler method.....	172
Improved Euler method.....	173
Runge-Kutta method	175
Predictor-Corrector methods	178
Derivation of an open formula of fourth-order.....	179
Derivation of a closed formula of fourth-order	180
Estimation of truncation error	181
Hamming's Open formula.....	182
Hamming's Closed formula	183
Modifier for the Adams formula	183
Higher-order initial value problems.....	186
Sets of coupled first-order differential equations.....	189
Sets of coupled higher-order differential equations	190
Boundary value problems	195
Shooting method	195
References	200
 Chapter 3.7 Solution of Partial Differential Equations	 201
Generalization of the method for any shape	205
Initial and Boundary conditions.....	206
Dirichlet conditions.....	206
Neumann conditions.....	207
Robbins conditions.....	207

Calculation of average temperature	210
Fully implicit method.....	212
Equations at the boundaries	213
Derivative boundary with an imaginary node	214
Crank-Nicolson method	219
Development of a general expression	220
Development of computer programs	221
Development of a general program.....	224
Alternating approach.....	226
Solution of a two-dimensional PDE	228
Temperature history in a two-dimensional slab	229
References	235
 Chapter 3.8 Errors	 237
Absolute error.....	237
Relative error	238
Inherent error	238
Truncation error.....	239
Roundoff error	239
Error estimation	240
References	242
 Chapter 3.9 Finite Element Methods	 243
Procedure steps.....	244
Discretization of the domain.....	244
Development of element equations	244
Assembly of element equations	245
Properties of elements	245
One-dimensional simplex element	245
Two-dimensional simplex element.....	247
Embedding element equations	252
References	261
 PART IV. DIMENSIONAL ANALYSIS	 263
Methods of Dimensional Analysis	265
Method 1	265
Pi Theorem of Buckingham.....	275

Method 2.....	276
Model Testing for Scaling Up.....	282
Geometric similarity.....	282
Kinematic similarity	283
Dynamic similarity	283
Symbols	286
References.....	288
PART V. APPLICATIONS	289
Air Psychrometrics.....	290
Psychrometric relations.....	292
Calculation of the psychrometric properties	294
Tubular Heat Exchanger with Annular Fin	299
Formulation of the problem	301
Input data	302
Heat Transfer Coefficient in Puffing of Rice	303
Formulation of the problem	304
Input data	306
Residence Time of a Particle Inside a Cyclone Separator.....	306
Formulation of the problem	306
Input data	308
Drying Simulation using Diffusion and Thin-Layer	
Concepts	309
Formulation of the problem	310
Thin-Layer Equation.....	311
Diffusion equation	312
Fick's first law.....	312
Fick's second law	313
Coefficients of the thin-layer drying equation.....	316
Diffusion coefficient	316
Latent heat of evaporation	317
Equilibrium moisture content.....	317
Heat transfer coefficient.....	317
Specific product surface area	317
Input data	318
References	320

APPENDICES	321
Appendix A.3.1. Program listings for numerical differentiation.....	321
Appendix A.3.2. Program listings for calculating root(s)	329
Appendix A.3.3. Program listings for numerical integration	341
Appendix A.3.4. Program listings for solution of linear algebraic equations	351
Appendix A.3.5. List of programs used in Regression Analysis.....	359
Appendix A.3.6. Program listings for solutions of ODE	377
Appendix A.3.7. Program listings for solutions of PDE.....	409
Appendix A.5. Complete program listings for solutions of application problems.....	447
A.5.1. Psychrometric functions.....	447
A.5.2. Psychrometric properties.....	452
A.5.3. Thermal properties for the fin problem	467
A.5.4. Solution for the circular fin problem.....	467
A.5.5. Thermal properties for the puffing problem.....	472
A.5.6. Program segment for the coefficient values	474
A.5.7. Heat transfer coefficient for the puffing of rice.....	475
A.5.8. Cyclone separator.....	479
A.5.9. Program segment for user input	484
A.5.10. Concurrent flow drying simulation	486
INDEX	495

PART I

UNITS AND DIMENSIONS

Though the official international system of units is the SI system (Systemè International d'Unitès), older systems such as the Metric (CGS and MKS) and the British/Imperial (FPS) are still in use and probably will be around for some time. Thus a working knowledge in all these three systems is necessary and a conversion table is usually the solution to convert data from one system to another.

Most of the time when one is desperate to convert the data to another unit, either the table that has been so carefully kept aside is not found, or the table does not have the exact conversion needed at that moment. Many a time one table would give the conversion up to a fixed decimal place when one was interested in a higher accuracy. A little variation among the same conversion factor reported in different tables is also a commonplace.

There is an inherent danger in using conversion tables without understanding the related physical phenomenon. To avoid controversies and confusion, it has been attempted here to develop an effective method to make these conversions more meaningful and accurate. The reader has to understand some basic concepts/definitions, and utilize minimum information about numeric values of some parameters in order to use successfully the method described here. All these concepts, definitions, information, and the method have been discussed in this part.

The objective of this work is to help the interested readers to convert the data from one unit to another without being totally dependent on conversion tables. Also later while discussing dimensional analysis in part IV, the concept gathered here will be immensely helpful.

Concepts

The focus of this part is on the fundamental dimensions such as length, mass, time, and temperature and basic concepts of the derived dimensions such as force, work, energy, power, pressure, absolute viscosity, and kinematic viscosity. This will cover most of the engineering units (unit operations) used today in the thermo-mechanical area. The units used in electricity, magnetism, and sound are not discussed here.

Let us denote the four fundamental dimensions, namely, length, mass, time, and temperature by the single letters [L], [M], [T], and [θ], respectively. Every other derived dimension in the list above can be expressed by a combination of the first three fundamental dimensions.

The force is defined as the product of mass and acceleration $[MLT^{-2}]$, work and energy as the product of force and distance $[ML^2T^{-2}]$, power as the rate of doing work $[ML^2T^{-3}]$, pressure as the force per unit area $[ML^{-1}T^{-2}]$, absolute viscosity as the shearing stress per unit change of velocity of fluid flow in thin layer over a surface with respect to the layer thickness $[ML^{-1}T^{-1}]$, and kinematic viscosity as the ratio between absolute viscosity and the density of the fluid $[L^2T^{-1}]$. In Table 1.1, all these basic and derived dimensions are shown along with corresponding units in all the three unit systems. The basic definitions of the derived units shown in Table 1.1 by asterisks are Newton [N], Pascal [Pa], Joule [J], and Watt [W] in SI system, dyne [dyn], [erg], calorie [cal], Poise [P], and Stoke [St] in CGS system, and poundal [pdl] in FPS system in terms of the first three basic dimensions.

Besides the above-mentioned basic dimensions and concepts, the following numbers should be memorized up to the last figure. These numbers are values of acceleration due to gravity, g ($=9.80665 \text{ m/s}^2$), specific gravity of water ($=1.0$ at 4°C) and that of mercury ($=13.595098$ at 0°C), the constant pi [π] the value of which up to ninth decimal place is 3.141592654, and the prefixes used only with metric and SI units as given in Table 1.2. The constant (π) is defined as the ratio between the circumference and the diameter of a circle. The specific gravity is defined as the mass of a substance divided by the mass of an equal volume of water at the same temperature.

Conversion factors between inch [in] and centimeter [cm], foot [ft] and [in], [ft] and yard [yd], [yd] and [mile], hectare [ha] and [m^2], [acre] and [m^2], bushel [bu] and [m^3], cm^3 [cc] and liter [l], pound [lb] and gram [g], pound [lb] and ounce [oz], metric ton [t] and kilogram [kg], minute [min] and second [s], hour [h] and [min], calorie [cal] and [erg], horsepower [hp] and the basic FPS unit for power [ft.lbf/s], atmosphere [atm] and [mm Hg], [Bar] and Pascal [Pa], [Torr] and [mm Hg], and [rad] and degree [$^\circ$] are given in Table 1.3.

The basic exact conversions given in Table 1.3 should be remembered. Units such as [ha], [acre], [bu] have been included there, keeping the people linked with agriculture in mind. Those who will be needing other relevant

Table 1.1. Units and dimensions in different unit systems

Quantities [Dimensions]	CGS [MKS]	FPS (Imperial/English)	SI (International)
Mass [M]	g [kg]	lb	kg
Length [L]	cm [m]	ft	m
Time [T]	s	s	s
Temperature [θ]	$^{\circ}\text{C}$ [K]	$^{\circ}\text{F}$ [$^{\circ}\text{R}$]	$^{\circ}\text{C}$ [K]
Force [MLT^{-2}]	g.cm/s^2 [dyn]* kg.m/s^2 [kgf]	lb.ft/s^2 [lbf] [lb.ft/s^2 (pdl)]*	kg.m/s^2 [N]*
Work [ML^2T^{-2}]	dyn.cm [erg]* [kgf.m]	ft.lbf	N.m
Energy [ML^2T^{-2}]	erg [cal] [kgf.m]	ft.lbf [Btu]	N.m [J]*
Power [ML^2T^{-3}]	erg/s [kgf.m/s]	ft.lbf/s [hp]	J/s [W]*
Pressure [$\text{ML}^{-1}\text{T}^{-2}$]	cm water [kgf/m 2]	lbf/in 2 [psi]	N/m 2 [Pa]*
Abs. viscosity [$\text{ML}^{-1}\text{T}^{-1}$]	g/cm.s [P]* [kg/m.s]	lb/ft.s	Pa.s
Kin. viscosity [L^2T^{-1}]	cm^2/s (St)* [m 2 /s]	ft 2 /s	m 2 /s

* by definition.

Table 1.2. Prefixes in metric and SI units

Prefix	Symbol	Multiple	Prefix	Symbol	Multiple
exa	E	10^{18}	deci	d	10^{-1}
peta	P	10^{15}	centi	c	10^{-2}
tera	T	10^{12}	milli	m	10^{-3}
giga	G	10^9	micro	μ	10^{-6}
mega	M	10^6	nano	n	10^{-9}
kilo	k	10^3	pico	p	10^{-12}
hecto	h	10^2	femto	f	10^{-15}
deka	da	10^1	atto	a	10^{-18}

Table 1.3. Basic exact conversions

To convert from	To	Multiply by
in	cm	2.54
ft	in	12
yd	ft	3
mile	yd	1760
acre	ft ²	43560
ha	m ²	10 ⁴
bu	m ³	0.03524
l	cc	1000
lb	g	453.59237
lb	oz	16
ton	kg	1000
min	s	60
h	min	60
cal*	erg	4.1868x10 ⁷
cal@	erg	4.1840x10 ⁷
hp	ft.lbf/s	550
Btu/lb. °F	cal/g. °C	1
atm	mm Hg	760
Bar	Pa	10 ⁵
Torr	mm Hg	1
rad	degree	180/π

* International Steam Table; @ Thermochemical

basic units in their fields can easily add those in the list expressing them in terms of the four fundamental dimensions, namely, [M], [L], [T] and [θ].

The English unit [slug] is not included in Table 1.3. This is a special fundamental unit in the English system for force. Later it was related with [lbf] and the unit of acceleration in FPS system. The Newton's proportionality factor, g_c has a direct bearing with [slug]. It is mainly used to balance units among different terms of the same equation, e.g., energy balance equation of Bernoulli in fluid flow. This factor is unity when expressed in fundamental units. Slug and g_c bear the following derived units:

$$1 \text{ slug} = 1 \text{ lbf.s}^2/\text{ft}$$

and

$$g_c = 32.1740485564 \text{ lb/slug}$$

Concepts such as implicit relationship between British thermal unit [Btu] and [cal] (McCabe et al., 1985), explicit relations between manometric height and pressure, kilogram force [kgf] and [kg.m/s²], pound force [lbf] and

[lb.ft/s²], and the relation between temperature difference in different unit systems are necessary to be understood.

Consider the following implicit relationship,

$$1 \text{ Btu/lb.}^\circ\text{F} = 1 \text{ cal/g.}^\circ\text{C}$$

From this relationship, the conversion factor between [Btu] and [cal], defined by the International Steam Table, can be calculated if the left hand side of the equation is converted to [Btu/g.°C]. This conversion with some others are given later while explaining the method.

To convert pressure in manometric column height of any specified liquid to any other unit, the following concept should be understood. Let us consider a U-tube manometer with a pressure difference of a column height of H of a liquid. The internal area of cross section of U-tube is A_i and the density of the liquid inside is denoted by ρ . The mass of the liquid above the common level (of height H), m_L is given by

$$\begin{aligned} m_L &= (\text{volume of the column}).(\text{density of liquid}) \\ &= (A_i H) (\rho) \end{aligned}$$

Force exerted by this column of liquid

$$\begin{aligned} &= (\text{mass}).(\text{acceleration due to gravity}) \\ &= (A_i H \rho) (g) \end{aligned}$$

Pressure due to this column at the common level

$$\begin{aligned} &= (\text{Force})/(\text{Area}) \\ &= (A_i H \rho g)/(A_i) \\ &= H \rho g \end{aligned}$$

This also explains an important fact — that the dimension of the U-tube does not have any influence on the measurement of pressure, i.e., irrespective of the U-tube diameter, the difference of column height will continue to remain as H.

The pound force [lbf] is defined as the product of mass in [lb] and acceleration due to gravity in [ft/s²]. Similarly, the kilogram force [kgf] is the product of mass in [kg] and acceleration due to gravity in [m/s²].

$$1 \text{ lbf} = 32.1740485564 \text{ lb.ft/s}^2 \text{ and } 1 \text{ kgf} = 9.80665 \text{ kg.m/s}^2$$

The relationships to convert degree Fahrenheit [°F] to degree Celsius [°C], [°C] to Kelvin [K], and [°F] to degree Rankine [°R] are as follows:

$$^{\circ}\text{C} = (^{\circ}\text{F} - 32)/1.8$$

$$\text{K} = ^{\circ}\text{C} + 273.15; \text{ and } ^{\circ}\text{R} = ^{\circ}\text{F} + 459.69$$

Any other relationships between K, $^{\circ}\text{R}$, $^{\circ}\text{C}$, and $^{\circ}\text{F}$ can be derived from the above.

When the difference in temperature is in question, as in the case of derived units of specific heat [$\text{L}^2\text{T}^{-2}\theta^{-1}$], thermal conductivity [$\text{MLT}^{-3}\theta^{-1}$], and heat transfer coefficient [$\text{MT}^{-3}\theta^{-1}$], degree Celsius can be replaced by Kelvin and vice versa and degree Fahrenheit by degree Rankine and vice versa. The reason follows:

$$\text{K}_1 = ^{\circ}\text{C}_1 + 273.15 \quad \text{and} \quad \text{K}_2 = ^{\circ}\text{C}_2 + 273.15$$

$$\text{K}_1 - \text{K}_2 = \Delta\text{K} = ^{\circ}\text{C}_1 - ^{\circ}\text{C}_2 = \Delta^{\circ}\text{C}$$

Similarly,

$$\Delta^{\circ}\text{R} = \Delta^{\circ}\text{F}$$

Also, if the relationship to convert degree Fahrenheit to degree Celsius is considered (the reverse of which can be easily derived), relationships between temperature difference in any unit system can be established in the following manner:

$$^{\circ}\text{C}_1 = (^{\circ}\text{F}_1 - 32)/1.8 \quad \text{and} \quad ^{\circ}\text{C}_2 = (^{\circ}\text{F}_2 - 32)/1.8$$

$$^{\circ}\text{C}_1 - ^{\circ}\text{C}_2 = \Delta^{\circ}\text{C} = (^{\circ}\text{F}_1 - ^{\circ}\text{F}_2)/1.8 = \Delta^{\circ}\text{F}/1.8 = \Delta\text{K}$$

and

$$\Delta^{\circ}\text{F} = 1.8 \Delta^{\circ}\text{C} = \Delta^{\circ}\text{R}$$

The method to convert any derived unit to another requires first to split the derived unit into fundamental units as per definition. As an example, if it is asked to convert 1 erg to SI unit, the definition of [erg] should be recalled. It is defined as [$\text{g}\cdot\text{cm}^2/\text{s}^2$], where [g], [cm], and [s] are the three fundamental units. Now, since it has been asked to convert to SI system, the basic units in CGS system, in this case [g] and [cm], should be converted to corresponding units of SI, namely, [kg] and [m]. The unit of time being same in all the three unit systems, it can be kept undisturbed.

Example 1.1. Convert [erg] to the equivalent unit in SI.

$$\begin{aligned}
 \text{Solution. } 1 \text{ erg} &= 1 \text{ g.cm}^2/\text{s}^2 \\
 &= 1 [\text{g}] \times 10^{-3} [\text{kg/g}] \cdot [\text{cm}^2] \times 10^{-4} [\text{m/cm}]^2/\text{s}^2 \\
 &= 10^{-7} \text{ kg.m}^2/\text{s}^2 \\
 &= 10^{-7} \text{ J} \quad [1 \text{ kg.m}^2/\text{s}^2 = 1 \text{ J}]
 \end{aligned}$$

It can be observed from the definitions of derived units in SI that they are directly obtained from the corresponding derived dimensions. This simplicity makes this system acceptable unanimously throughout the world.

Once the derived units are written in fundamental units, for each of the units, it should be asked what is the value of the goal unit per corresponding unit to be converted. In the previous example, [g] is the unit to be converted, and [kg] is the corresponding goal unit. It is thus asked: how many [kg] per 1 g, and it is 10^{-3} , similarly for the next unit [cm^2], how many [m^2] per 1 cm^2 is the question, and the answer is 10^{-4} .

Example 1.2. Convert [Btu] to [cal].

Solution. Let us consider the implicit relationship:

$$1 \text{ Btu/lb.}^\circ\text{F} = 1 \text{ cal/g.}^\circ\text{C}$$

$$\begin{aligned}
 \text{Now, } 1 \text{ Btu/lb.}^\circ\text{F} &= 1 \text{ Btu} / [\text{lb}] \times 453.59237 [\text{g/lb}] \cdot [^\circ\text{F}] \times (1/1.8) [^\circ\text{C}/^\circ\text{F}] \\
 &= 3.9683207193 \times 10^{-3} \text{ Btu/g.}^\circ\text{C} \\
 &= 1 \text{ cal/g.}^\circ\text{C}
 \end{aligned}$$

$$\therefore 1 \text{ Btu} = 251.9957611 \text{ cal}$$

During all these conversions, it should be borne in mind that calculation in the conversion of each basic unit should not be done separately. The numbers may get approximated by truncation and round off errors if so done, and the end result may deviate from a more exact value.

Another important fact should be made clear at this point — that it is not necessary to remember values of the same parameter in different units. Let us take an example of the acceleration due to gravity, g. In the text it has been given in metric or SI unit, because the value is exact. Now, its value in FPS unit may not be remembered, rather the same may be calculated each time it is needed.

$$\begin{aligned}
 9.80665 \text{ m/s}^2 &= 9.80665 [\text{m}] \times (1/0.3048) [\text{ft/m}]/\text{s}^2 \\
 &= 32.1740485564 \text{ ft/s}^2
 \end{aligned}$$

8 *Applied Numerical Methods for Food and Agricultural Engineers*

Using this method, the numeric values of [slug] and g_c can easily be established when both are expressed in their corresponding fundamental units.

$$\begin{aligned}
 1 \text{ slug} &= 1 \text{ lbf} \cdot \text{s}^2/\text{ft} \\
 &= 32.1740485564 [\text{ft}/\text{s}^2] \cdot [\text{lb}] \cdot [\text{s}^2]/[\text{ft}] \\
 &= 32.1740485564 \text{ lb} \\
 &= 32.1740485564 [\text{lb}] \times 0.45359237 [\text{kg}/\text{lb}] \\
 &= 14.5939029372 \text{ kg}
 \end{aligned}$$

and

$$\begin{aligned}
 g_c &= 32.1740485564 \text{ lb/slug} \\
 &= 32.1740485564 [\text{lb}]/[\text{slug}] / 32.1740485564 [\text{lb/slug}] \\
 &= 1
 \end{aligned}$$

Thus, it is observed that the numeric value of this factor is unity when the same is expressed in fundamental units.

The third example is chosen to show how the unit of temperature is converted from one system to another when it is embedded in a derived unit. Here it should be made clear that, in such a case, the concept of temperature interval will have to be applied. If we would like to convert 1 Btu/h.ft.°F to SI unit, the implicit definition should be recalled. Since we have already converted [Btu] to [cal], we will use it directly without repetition.

Example 1.3. Convert [Btu/h.ft.°F] to the equivalent SI unit.

Solution. 1 Btu/h.ft.°F

$$\begin{aligned}
 &= 1 [\text{Btu}] \times 251.9957611 [\text{cal}/\text{Btu}] / [\text{h}] \times 3600 [\text{s}/\text{h}] \\
 &\quad [\text{ft}] \times 0.3048 [\text{m}/\text{ft}] \cdot [^\circ\text{F}] \times (1/1.8) [^\circ\text{C}/^\circ\text{F}] \\
 &= 413.378873196 \times 10^{-3} \text{ cal/s.m.}^\circ\text{C}
 \end{aligned}$$

Now, 1 cal = 4.1868×10^7 erg [International Steam Table]

$$\begin{aligned}
 &= 4.1868 \times 10^7 [\text{erg}] \times 10^{-7} [\text{J}/\text{erg}] \quad [1 \text{ erg} = 10^{-7} \text{ J}] \\
 &= 4.1868 \text{ J}
 \end{aligned}$$

$$\begin{aligned}
 1 \text{ Btu/h.ft.}^\circ\text{F} &= 413.378873196 \times 10^{-3} \text{ cal/s.m.}^\circ\text{C} \\
 &= 413.378873196 \times 10^{-3} [\text{cal}] \times 4.1868 [\text{J}/\text{cal}] / \text{s.m.}^\circ\text{C} \\
 &= 1.7307346663 \text{ J/s.m.}^\circ\text{C} \\
 &= 1.7307346663 \text{ W/m.}^\circ\text{C} \quad [1 \text{ J/s} = 1 \text{ W}]
 \end{aligned}$$

Example 1.4. Convert [cm water] to SI unit of pressure.

Solution. $1 \text{ cm water} = 1 [\cancel{\text{cm}}] \times 10^{-2} [\text{m}/\cancel{\text{cm}}] \text{ water}$
 $= 10^{-2} \text{ m water}$
 $= 10^{-2} [\text{m}] \times 999.07156 [\text{kg}/\text{m}^3] \times 9.80665 [\text{m}/\text{s}^2]$
 $= 97.9754512 \text{ kg}/\text{m} \cdot \text{s}^2$
 $= 97.9754512 \text{ Pa} \quad [1 \text{ kg}/\text{m} \cdot \text{s}^2 = 1 \text{ Pa}]$

The value for the density of water at 60°F was taken in FPS unit from McCabe et al., 1985, and it was converted to SI unit using the present method.

The final conversion explained is a little uncommon and may not be found in many conversion tables. It is attempted to show the justification and convenience of the method described here.

Example 1.5. Convert [Btu.in/min.ft².°R] to SI unit.

Solution. $1 \text{ Btu.in}/\text{min.ft}^2 \cdot ^\circ\text{R}$
 $= 1 [\cancel{\text{Btu}}] \times 251.9957611 [\text{cal}/\cancel{\text{Btu}}] \times 4.1868 [\text{J}/\text{cal}]$
 $\quad \cdot [\cancel{\text{in}}] \times 0.0254 [\text{m}/\cancel{\text{in}}] / [\cancel{\text{min}}] \times 60 [\text{s}/\cancel{\text{min}}]$
 $\quad \cdot [\cancel{\text{ft}^2}] \times 0.3048^2 [\text{m}^2/\cancel{\text{ft}^2}] \cdot [^\circ\text{R}] \times (1/1.8) [\text{K}/^\circ\text{R}]$
 $= 8.6536733315 \text{ J} \cdot \text{m}/\text{s} \cdot \text{m}^2 \cdot \text{K}$
 $= 8.6536733315 \text{ W}/\text{m} \cdot \text{K}$

Symbols

A_i	Inside area of cross section of manometer tube, m^2
g	Acceleration due to gravity, m/s^2
g_c	Newton's proportionality factor, lb/slug
H	Manometric column height of liquid, m
L	Length, m
M	Mass, kg
m_L	Mass of a column of liquid of height H , kg
T	Time, s
Δ	Small amount of/Difference of
ρ	Density, kg/m^3
θ	Temperature, $^\circ\text{C}$ or $^\circ\text{F}$ or $^\circ\text{R}$ or K

References

1. **ASAE EP285.7**, **ASAE Engineering Practice**, Use of SI (Metric) Units. *Transactions of the ASAE*, Vol.35(1), 366–373, 1992.
2. **Chandra, P. K.**, Unit Systems Revisited from Basic-Concept Angle. ASAE Paper No. 92-6547. St. Joseph, MI 49085 - 9659, 1992.
3. **Henderson, S. M. and Perry, R. L.**, *Agricultural Process Engineering*. The AVI Publishing Company, Inc., Westport, Connecticut, 1981.
4. **McCabe L. M., Smith, J. C. and Harriott, P.**, *Unit Operations of Chemical Engineering*. McGraw-Hill Book Co., New York, 1985.
5. **Mohsenin, N. N.**, *Thermal Properties of Foods and Other Agricultural Materials*. Gordon and Breach Science Publishers, Inc., London, 1980.
6. **Perry, R. H. and Chilton, C. H.**, *Chemical Engineers' Handbook*. 3rd Edition. McGraw-Hill Book Co., New York, 1973.
7. **Pitts, D. R. and Sissom, L. E.**, *Theory and Problems of Heat Transfer*. Schaum's Outline Series, McGraw-Hill Book Co., New York, 1977.
8. **Singh, R. P. and Heldman, D. R.**, *Introduction to Food Engineering*. Academic Press, Inc., Orlando, Florida, 306pp., 1984.

PART II

A CONCISE TREATISE ON FORTRAN

It is not the purpose of this book to write another exhaustive text on FORTRAN (**FOR**mula **TRAN**slator). The emphasis here is to present FORTRAN language with the purpose of getting the reader ready for using numerical methods explained in the remaining parts of this book. The main purpose of this book is to help understand the subject of numerical methods and apply the same in developing and solving mathematical models of physical processes pertinent to the area of food engineering. Without the knowledge of a proper computer language learning numerical methods does not make any sense. The proper computer language for this purpose is inevitably FORTRAN and will remain as one in the future in spite of recent mushrooming of myriad computer languages.

If one is organized, he or she can use FORTRAN as any other modern, disciplined, and structured computer language. The problem is that FORTRAN gives too much freedom to its users regarding its use and abuse, and, as a consequence, it suffers from being considered an undisciplined language. The fault is not in the language, rather it is with its irresponsible and slack users. There are other languages that, by their inherent structures, compel their users to be disciplined. In any case when it comes to complicated calculations involving matrices with large arrays, no other language comes close to FORTRAN with respect to its efficiency, minimal memory requirement and ease of programming.

The important part of computer programming is not just learning the various elements of the computer language but combining the commands or instructions into a logical sequence called a program to solve a problem

correctly and efficiently. Learning the elements of a computer language is easy. The part that counts is how to combine these elements into an efficient program.

In this part the focus will be on this second aspect. First, a simplest but complete program will be written and gradually the same will be step- wise improved, learning at each step various commands pertinent to the FORTRAN language. It will be taken for granted that the readers already are acquainted with the common computer terms, such as software, hardware, machine language, CPU, compilers, etc.

Before starting to write a program, it is necessary to understand the concepts of (i) number systems, (ii) structured programming, (iii) variables and constants, and (iv) arithmetic assignment statements and to learn about (i) operation symbols and (ii) coding FORTRAN statements.

Number systems

The numbering system we are accustomed to is based on ten digits, namely, 0,1,2,3,4...9 and the digits/numbers are known as decimal numbers. In this system the base is the number 10. Thus, in a decimal number 110100, each digit contributes something to the final answer. To determine the contribution of each digit, the same is multiplied by some power of the base, which is 10 in decimal numbering system. This power is a function of the position where the digit appears in the string. Then the individual contributions are added up to arrive at the result in decimal system. Let the decimal number 110100 be expressed as 110100_{10} to differentiate it from the same string in other numbering systems. The power for the last digit on the right hand side starts with zero and is incremented by one for each of the remaining digits starting from right to left of the number. The individual contributions can be expressed as below:

$$\begin{array}{c}
 110100_{10} = 0 \times 10^0 + 0 \times 10^1 + 1 \times 10^2 + 0 \times 10^3 + 1 \times 10^4 + 1 \times 10^5 \\
 \begin{array}{c}
 \text{Diagram showing brackets connecting each digit to its corresponding power of 10:} \\
 \text{Digit 0 (rightmost) to } 10^0, \text{ Digit 0 to } 10^1, \text{ Digit 1 to } 10^2, \text{ Digit 0 to } 10^3, \text{ Digit 1 to } 10^4, \text{ Digit 1 to } 10^5.
 \end{array} \\
 = 0 + 0 + 100 + 0 + 10000 + 100000 \\
 = (110100)_{10}
 \end{array}$$

As opposed to the conventional decimal system, computers work on a system of just two digits, namely 0 and 1. It is called a **binary number system** and the digits 0 and 1 are known as **bits**, the abbreviation of **binary**

digits. All alphabetic information, such as letters and symbols and decimal quantities, is expressed as the combinations of 0 and 1 called **bytes**. In an 8-bit computer, the capital **A** is expressed by 01000001 whereas the lower case **a** by 01100001. Thus the byte for **A** is not the same as that for **a**. The binary number 110100_2 having a base of 2 should not be confused with the decimal number 110100_{10} . This binary number can be converted to a decimal number in the similar fashion described earlier the only difference being that the base here is 2 instead of 10:

$$\begin{aligned} 110100_2 &= 0 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 \\ &= 52_{10} \end{aligned}$$

There are other numbering systems, such as, **hex** having a base of 6 and **octal** with a base of 8.

Structured programming

An efficient and structured style in writing programs in FORTRAN has been already mentioned. Most often computer programs are not documented enough to convey their functioning clearly to the users, contain statements that are not universal to various FORTRAN compilers, do not meet user requirements, and are difficult to update to meet new demands. A student should not merely learn to code FORTRAN statements, he or she should also learn how these statements can be put together to turn the program into a high-quality one that is easy to understand, efficient in using computer resources, and easy to incorporate changes into when and where necessary.

In structured programming, the objective should be to code an accurate, error-resistant, and maintainable program in a straightforward (sequential) and easily understood manner. All the computer programs can be written using only three logic structures or any combinations of them:

1. Simple sequence
2. Selection
3. Repetition

In a disciplined programming, these structures are useful because the program is simplified. There is always a single point entry into the structure and a single point exit from it. These also help read the program from top to bottom, thus making the logic of the program more visible for checking.

The simple sequence structure consists of one action followed by another from top to bottom of a program.

The selection structure consists of a test for a condition followed by more than one alternative paths for the program to follow. After completing one of the paths, the program returns to a single point at the end of the structure.

In the repetition structure, an operation or a group of operations is repeated until some condition(s) is/are satisfied. This repetition structure is called a **loop**. The program logic tests a condition governing the continued operation of the loop. If the condition is satisfied, the program executes the operation and loops back for another test. When the condition ceases to be true, the repetition stops.

As discussed earlier, the FORTRAN language was not designed for structured programming, but it is possible and highly desirable to follow the ideas. Throughout the present part these ideas will be pursued.

Variables and constants

In FORTRAN, constants and variables bear the same sense as in mathematical notation. An unknown quantity or a quantity that can vary in different situations is a variable and is assigned a variable name. This name refers to a computer memory location where the data is stored. It may be a single letter symbol, such as A or it may be a descriptive name, such as ADD. In FORTRAN, a number written in an arithmetic expression is called a constant. A constant can be given a name exactly like a variable, but, unlike the variable, it remains at a single value throughout the program.

In some computations, it may become necessary that certain values be only integers (1,3,10,1000, etc.) while the others be numbers with fractional parts (1.0, 3.5, 10.45, 1000.8765, etc.). These are referred to as integers and real numbers, respectively. In FORTRAN, there are two different types of variables and constants to distinguish between integer data and real or floating point data.

An integer quantity, which is also known as a fixed point quantity, is one that does not have any decimal parts. It contains only whole numbers. In FORTRAN, any variable name starting with or constituting one of the six letters I, J, K, L, M, and N represents whole numbers. The name of a variable starting with or constituting any letter other than the above six makes it real. The internal representation of an integer number is exact, whereas that of a real data is a processor-dependent approximation, which may not be exact for some values of real data. The FORTRAN compiler treats integer and real data differently.

At this point the rules for variable names should be mentioned. For any type of variable, (i) its first character must be alphabetic, (ii) no special characters are allowed, that is, names can contain only letters and numbers, but not any symbols, such as \$, @, ?, &, #, !, etc., and (iii) the total number of characters must not exceed six. If it does, the variable name will be automatically truncated after the sixth character. Thus the FORTRAN compiler will not differentiate between the variables TAYLOR and TAYLOR1, because the latter will be truncated to TAYLOR at the time of compilation.

There are four additional types of data that are allowed in FORTRAN. These are as follows:

1. Character data
2. Double precision data
3. Complex data
4. Logical data

Character data items store alphabetic, numeric, and special characters in an internal representation that codes each character separately. A name should be stored as characters. A variable name referred to the character data is defined as character type. Numeric data defined as character type are stored as individual characters and cannot be used directly in arithmetic operations.

Double precision data items are real data that normally provide double precision of that the real variables otherwise use. This requires two storage spaces for each double precision data item. A single variable name, which is defined as double precision type, uses the double-length representation of its numeric value. For some FORTRAN compilers that use fairly limited precision, sometimes double precision is needed for computation involving very large or very small numbers.

Complex data items are represented by an ordered pair of real data items, one representing the real part and the other the imaginary part.

Logical data variables assume only values for **TRUE** or **FALSE**. If X is defined as a logical variable, X = **.TRUE.** will set X to the value for true.

Operation symbols

The operations that a computer is capable of performing can be grouped into the following four:

1. Input/output (**READ** and **WRITE**)
2. Arithmetic
3. Logic/Control (**IF**, **GOTO** etc.)
4. Specification (**FORMAT**)

This list identifies all the operations that a computer can perform. Any complicated program can be resolved into some combination of these four basic operations. Items 1, 3, and 4 will be explained later. In item 2, for arithmetic operations, the following operation symbols are used in FORTRAN:

<Addition>	+
<Subtraction>	-
<Division>	/
<Multiplication>	*
<Exponentiation>	**

Arithmetic assignments

An arithmetic assignment statement is an equation where a single variable is expressed as a function of some other variable(s) by an expression using one or more of the operation symbols given above. The right hand side of the equation should explicitly define how the basic arithmetic capabilities of the computer are to be combined to determine a desired value to be attributed to the variable on the left hand side of the equation. The single variable appearing on the left hand side of the equation tells where in memory the single numerical value, thus obtained, should be stored.

The purpose of any arithmetic statement is to express an equation in a proper and correct form to the computer. It should appear on one continuous line while the same equation may appear in more than one lines.

$$\frac{5(x-1)}{3x+4}$$

$$\frac{\quad}{6x^2}$$

Arithmetic statement

$$5.0*(X - 1.0)/((3.0*X + 4.0)/(6.0*X^2))$$

As observed in the above example, the use of parentheses becomes a necessity in the arithmetic statement to express the equation in a single line. Now, in order to express an equation as an arithmetic statement, it is necessary to understand precisely how the compiler will give preference to some operations while keeping all the rest in the waiting list.

FORTRAN uses both the precedence rule and parentheses rule to handle an arithmetic expression. The precedence rule states that all the exponentiations will be performed first starting from right to left in case of consecutive exponentiation operators [ex. $Y = X^{**2**3}$ is equivalent to X^{**8}], all multiplication and division next, and all addition and subtraction

last. The operation will be from left to right when precedence of operations is the same with the exception of exponentiation [ex. $Y = 20.0*6.0/5.0*10.0+2.0**2**3$ is equivalent to $Y = 120.0/5.0*10.0 + 2.0**8 = 240.0 + 256.0 = 496.0$]. When the exponent is a whole number (integer), it should be left like that without making it real. Thus $X**3$ should not be written as $X**3.0$. On the other hand, a real number without any decimal part should be written with a zero after the decimal point. For example, the real (floating point) number ten should be written as 10.0 and neither as 10., nor as just 10 — it is a matter of good habit, and the habit is formed at the time of learning. A bad habit formed at the time of learning is difficult to shake off later.

The parentheses rule states that operations in the innermost set of parentheses should be performed first and then in the next set, etc., until all the operations inside parentheses have been performed overruling the precedence rule. Then the remaining operations will be carried out according to the precedence rule. The example below explains these operations:

$$\begin{aligned}
 Y &= 20.0*6.0/5.0*10.0+2.0**2**3 \\
 &= ((20.0*6.0)/5.0)*10.0 + 2.0**(\underbrace{2**3}) \\
 &= (120.0/5.0)*10.0 + 2.0**8 \\
 &= 24.0*10.0 + 256.0 \\
 &= 496.0
 \end{aligned}$$

It is a good habit to use parentheses freely in order to improve readability of the statement rather than relying on the precedence rule. Whenever there is any doubt, parentheses rule should come in the rescue.

The rules for mixing integer and real types in an expression are very important in FORTRAN. These are explained below:

1. In case of such mixing, integer variables or values will be converted to real numbers.
2. In an equation, the type of the variable on its left hand side will predominate. If it is an integer, the fractional part of the value on the right hand side will be truncated.
3. Fractional parts from division of integer variables are truncated.
4. An integer variable/value to a real exponent will convert the result to a real value.
5. The combinations of types in exponentiation allowed in FORTRAN are as follows: (i) integer variable/value with integer exponent, (ii) real variable/value with real exponent, and (iii) real variable/value with integer exponent. The exponent may be any arithmetic expression.

Coding FORTRAN statements

When entering FORTRAN statements as a part of a program via a video terminal, as in the case of a microcomputer, it should be remembered that the screen has 80 columns and that the first 72 columns of it can be used.

The first column is used for comments. The letter **C** in the first column indicates that what follows on the line is not translated by the FORTRAN compiler. The line is printed as a part of the program listing. It is used for explanatory comments.

The first to fifth columns are used for statement reference number. This number can be placed anywhere within these five columns.

The sixth column is reserved for putting a single number (0 to 9) or an asterisk (*) to indicate that the line is a continuation of the previous line(s). It is used when the FORTRAN statement is too long to be accommodated within 72 columns.

Columns 7 to 72 are used for FORTRAN statement. The statement can begin anywhere between this limit.

Programming in FORTRAN

First a program will be written for calculating two real roots of a quadratic equation. To do this, there should be a previous knowledge about analytical expressions for the two roots. The quadratic equation is expressed as

$$y = f(x) = ax^2 + bx + c$$

The roots of this quadratic (second power in x) equation are, say, r_1 and r_2 , which, if substituted in the equation, the numeric value of the function $f(x)$ will be zero, or in other words,

$$f(r_1) = f(r_2) = 0$$

The analytical expressions for r_1 and r_2 are as follows:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

and

$$r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

In FORTRAN statements these can be written as

$$R1 = (-B + (B**2 - 4.0*A*C)**0.5)/2.0/A \quad 2.1$$

$$R2 = (-B - (B**2 - 4.0*A*C)**0.5)/2.0/A \quad 2.2$$

where,

A, B, and C : coefficients of the quadratic equation (real variables in FORTRAN)

In Statements (2.1) and (2.2), the term $(B**2 - 4.0*A*C)$ is common. If it is denoted by D, we can rewrite these statements as

$$R1 = (-B + \text{SQRT}(D))/2.0/A$$

and

$$R2 = (-B - \text{SQRT}(D))/2.0/A$$

Here one keyword in FORTRAN is learned, which is **SQRT**, the abbreviation of **S**quare **R**oot. It is an intrinsic function in FORTRAN. In the course of developing and improving a program such keywords will be gradually learned. The keywords in FORTRAN are reserved words and should not be used as variable names. A list of such reserved words in FORTRAN is provided in Table 2.2 at the end of this part.

STOP and END statements

The two other keywords that are very important in a FORTRAN program and without which the program will not be complete are **STOP** and **END**. The **STOP** statement is used to specify that the program execution is to be terminated. There may be more than one **STOP** instructions in a large program. When the compiler encounters this keyword, it understands that the program is to be stopped and the program control to be directed to the last statement of the program, which is **END**. This keyword tells the compiler that there are no more statements in the program unit.

List-directed input and output

In order to supply the values of A, B and C from the screen/monitor (input) and write the results, that is, values of R1 and R2 on it (output), two keywords should be explained :

1. List-directed input (free format **READ**) and
2. List-directed output (free format **WRITE**)

The general input **READ** has the following syntax:

READ(n_1, n_2) $v_1, v_2, v_3, \dots, v_n$

where,

n_1 : unit number/symbol for input data source

n_2 : format statement number/symbol

$v_1, v_2, \dots, v_n$: variables to read

Also the output statement **WRITE** has the same syntax as that of **READ**. In either case, n_1 is a whole number or an asterisk. In case of **READ**, it specifies to the compiler the unit number of an external input file or the screen from where the values of the variables $v_1, v_2 \dots$, etc., are to be read. In case of **WRITE**, this specifies the external output file or the screen to where the values are to be written. When the source or the destination is the screen/monitor, an asterisk is usually used.

The format number, n_2 is also an integer and used only with **FORMAT** statement, which will be explained later. When no format is specified, an asterisk is usually used instead of a number. Thus the **FORTRAN** statement **READ**(*,*) A,B,C will instruct the compiler to read values of A, B, and C from the screen (the first asterisk in the argument list) and to read them in free format (the second asterisk). The values of A, B, and C can be provided from the screen with a space or comma between them. Similarly, **WRITE**(*,*) R1, R2 will write the values of R1 and R2 on the screen with the default format of the compiler.

READ(*, *) A,B,C

from monitor
values of A, B, C

with free format

WRITE(*, *) R1,R2

to monitor
values of R1,R2

with free format

With the knowledge gained so far the following program is written. This program is written with the intention of getting started with programming in FORTRAN. It is a small and complete program in accordance with the grammar of FORTRAN compiler, though it may have to go a long way to be really complete with respect to what is expected of such a program.

Example program 2.1

```
C _____
C
C  THIS PROGRAM CALCULATES TWO REAL ROOTS OF A QUADRATIC EQUATION
C
C  VALUES OF THE COEFFICIENTS ARE ASSIGNED TO VARIABLES A, B & C
```

```

C   AND VALUES OF THESE VARIABLES ARE READ FROM SCREEN
C
C       PROGRAM EXMP01
C
C       READ(*,*) A,B,C
C
C   ANOTHER VARIABLE D IS DEFINED TO HOLD THE VALUE OF  $B^2-4AC$ 
C
C        $D = B^2 - 4.0 * A * C$ 
C
C   REAL VARIABLES R1 AND R2 ARE USED TO ASSUME VALUES OF THE ROOTS
C
C        $R1 = (-B + \text{SQRT}(D))/2.0/A$ 
C        $R2 = (-B - \text{SQRT}(D))/2.0/A$ 
C
C   RESULTS ARE PRINTED ON THE SCREEN
C
C       WRITE(*,*) R1,R2
C
C   THE PROGRAM IS STOPPED AND TERMINATED
C
C       STOP
C       END
C

```

At the beginning of the program the first statement uses another keyword **PROGRAM** followed by the name **EXMP01**. This is a way to give a name to a program but is not compulsory. The file extension name **FOR** is needed for a FORTRAN file, without which the compiler will not recognize it. When this program EXMP01.FOR is compiled, no error is detected, and the executing program EXMP01.EXE is created. To run the program, the principal name of the program EXMP01 is typed and the key <ENTER> is pressed. The program runs, prints the values of R1 and R2 on the screen and terminates.

Since the compiler browses through the program from top to bottom, passing through each and every executable statement, unless it is asked to do otherwise, it passes first through the comments and ignores them. Then it encounters first executable FORTRAN statement **READ** and waits until the values of A, B, and C are typed on the video screen with a space or a comma between them. Next, it calculates the value of D and then values of R1 and R2 using the value of D calculated in the previous step. Finally it prints the values of R1 and R2 on the screen, and the program is terminated.

Let us now examine what items are lacking in this program and summarize them as follows:

(i) it is reading the coefficients only for one quadratic equation, (ii) it is not asking the user what data to type in, (iii) except for the numbers, it is not saying anything about the results, and (iv) it is not warning the user about the impossibility of calculating the square root of a negative number, D, before attempting such an operation.

Processing more than one datum

If there are more than one quadratic equations, it will be necessary to run the program all over again. It will be better if any desired number of sets of coefficients can be used without quitting the program. In order to use the same **READ** statement over and over again, a method called looping should be employed. To do this, two operations should be learned. The first is a statement to transfer control from one location of the program to the other. It is a **GO TO n** or **GOTO n** statement, where **n** is any statement reference number. A good practice is to start this number with 10 and the next with 20 and so on. The other is a simple **IF** statement to end the loop. This latter statement can be used in two ways to terminate a loop created by a **GOTO**.

Now let us introduce this new feature into the program that has just been written. Another **WRITE** statement before the **READ** will eliminate the second limitation mentioned before. The statement enclosed by two single apostrophes on the right hand side of **WRITE** statement will be exactly reproduced on the screen when the compiled program is run. Only the functional part of the program is listed in order to save space. Later the program will be rewritten in a more complete form.

Example program 2.2

```
C _____
C
10  WRITE(*,*) 'Give values of A, B and C'
    READ(*,*) A,B,C
C
    D = B**2 - 4.0*A*C
C
    R1 = (-B + SQRT(D))/2.0/A
    R2 = (-B - SQRT(D))/2.0/A
C
    WRITE(*,*) R1,R2
C
C    GO BACK TO READ MORE DATA ON A, B AND C
C
    GOTO 10
```

C

STOP

END

C

When the compiler sees **GOTO 10** statement, after doing one operation/loop, the program control is sent to **READ** statement having the reference number 10. The new values of A, B and C are again read, and the results are printed, and it continues repeating the loop without an end. It is called an endless loop situation, which should always be avoided.

A logical IF statement

A simple method for coding the end of such an endless loop is to use a logical **IF** statement, which tests if a required condition has been met and accordingly another **GOTO** statement transfers the control out of the loop. The logical **IF** statement has the following structure:

IF(relational expression) statement

Relational operators

The argument of **IF** may have one or more logical or relational expression(s). These expressions use relational operators to define a condition. Each of these operators are expressed by two letters with a period on either side, such as

.LT. : Less Than

.LE. : Less than or Equal to

.EQ. : EQual to

.NE. : Not Equal to

.GT. : Greater Than

.GE. : Greater than or Equal to

Logical operators

When two or more relational expressions are placed as arguments of **IF**, any or all of the following logical operators can be used,

.AND. : Both relations are true

.OR. : One or both relations are true

.NOT. : Opposite is true

Now, there are two ways to exit the infinite loop. The first one is to use a unique value of the variable in the input data, which may be zero or a negative number or a unique number that the regular input would not have. The other method is to establish a counter and keep track of the number of looping by incrementing the counter value by one at each loop. Let us include this feature in the program and check its functioning.

Example program 2.3

```

C _____
C
10  WRITE(*,*) 'Give values of A, B and C'
    READ(*,*) A,B,C
C
C    LET US CHECK IF A IS 9999. IF SO STOP EXECUTION.
C
    IF(A .EQ. 9999.0) STOP
C
    D = B**2 - 4.0*A*C
C
    R1 = (-B + SQRT(D))/2.0/A
    R2 = (-B - SQRT(D))/2.0/A
C
    WRITE(*,*) R1,R2
C
C    GO BACK TO READ MORE DATA ON A, B AND C
C
    GOTO 10
C
    STOP
    END
C _____

```

Thus the program stops when the value of A is given equal to 9999. In the second method, the counter is named as ICOUNT, and the initial value given is one before starting of the loop. Inside the loop it is incremented by one by the following statement:

$$\text{ICOUNT} = \text{ICOUNT} + 1$$

If there are 10 data sets to be read, the conditional statement **IF** is placed after the counter incremental step. Let us rewrite the program:

Example program 2.4

```

C
C
C      INITIALIZE THE COUNTER ICOUNT TO ONE
C
C          ICOUNT = 1
C
C      10  WRITE(*,*) 'Give values of A, B and C'
C          READ(*,*) A,B,C
C
C          D = B**2 - 4.0*A*C
C
C          R1 = (-B + SQRT(D))/2.0/A
C          R2 = (-B - SQRT(D))/2.0/A
C
C          WRITE(*,*) R1,R2
C
C      INCREMENT THE COUNTER VALUE BY ONE
C
C          ICOUNT = ICOUNT + 1
C
C      CHECK IF ICOUNT IS MORE THAN THE DATA SET NUMBER (=10)
C
C          IF(ICOUNT .GT. 10) STOP
C
C      GO BACK TO READ MORE DATA ON A, B AND C
C
C          GOTO 10
C
C          STOP
C          END
C

```

The second method eliminates one limitation of the previous method, which is that A can not have a value EQUAL TO 9999, and, hence, it is better and more general.

A program should be guarded against blowing up without any warning. A test that the value of D is negative can save the situation of attempting to calculate the square root of a negative number and blowing up in the process. Also, the value of A cannot be zero as it will attempt to divide by zero, which is not possible by any computer.

After reading the values of A, B and C an **IF** statement is necessary to test if D is negative and if A is zero. If it is so write an appropriate message transferring the control back to **READ**. The program now looks as follows:

Example program 2.5

```

C
C
C      INITIALIZE THE COUNTER ICOUNT TO ONE
C
C          ICOUNT = 1
C
C      10  WRITE(*,*) 'Give values of A, B and C'
C          READ(*,*) A,B,C
C
C          D = B**2 - 4.0*A*C
C
C      TEST IF D IS NEGATIVE OR A IS ZERO. IF NEGATIVE GO BACK
C      TO READ NEW VALUES OF A, B, C
C
C          IF((D .LT. 0.0) .OR. (A .EQ. 0.0)) GOTO 20
C          GOTO 30
C      20  WRITE(*,*) 'Change values of A, B & C'
C          GOTO 10
C      30  R1 = (-B + SQRT(D))/2.0/A
C          R2 = (-B - SQRT(D))/2.0/A
C
C          WRITE(*,*) R1,R2
C
C      INCREMENT THE COUNTER VALUE BY ONE
C
C          ICOUNT = ICOUNT + 1
C
C      CHECK IF ICOUNT IS MORE THAN THE DATA SET NUMBER (=10)
C
C          IF(ICOUNT .GT. 10) STOP
C
C      GO BACK TO READ MORE DATA ON A, B AND C
C
C          GOTO 10
C
C          STOP
C          END

```

Block IF statement

At this point it is important to remember structured programming. The program that has just been written has a total of four **GOTO** statements. The free use of **GOTO** makes the program difficult to read and inefficient. In structured programming the use of **GOTO** should be kept at minimum. A block-**IF** statement can eliminate the first two **GOTO**s, thus improving the readability of the program. The block-**IF** has the following structure:

```

      IF(condition) THEN
          statement(s)
      ELSE
          other statement(s)
      ENDIF

```

Introducing it in the program, it looks like this:

Example program 2.6

```

C _____
C
C      INITIALIZE THE COUNTER ICOUNT TO ONE
C
C          ICOUNT = 1
C
C      10  WRITE(*,*) 'Give values of A, B and C'
C          READ(*,*) A,B,C
C
C          D = B**2 - 4.0*A*C
C
C      TEST IF D IS NEGATIVE OR A IS ZERO. IF NEGATIVE GO BACK
C      TO READ NEW VALUES OF A, B, C
C
C          IF((D .LT. 0.0) .OR. (A .EQ. 0.0)) THEN
C              WRITE(*,*) 'Change values of A, B & C'
C              GOTO 10
C          ELSE
C              R1 = (-B + SQRT(D))/2.0/A
C              R2 = (-B - SQRT(D))/2.0/A
C          ENDIF
C
C      WRITE(*,*) R1,R2
C

```

```

C      INCREMENT THE COUNTER VALUE BY ONE
C
C      ICOUNT = ICOUNT + 1
C
C      CHECK IF ICOUNT IS MORE THAN THE DATA SET NUMBER (=10)
C
C      IF(ICOUNT .GT. 10) STOP
C
C      GO BACK TO READ MORE DATA ON A, B AND C
C
C      GOTO 10
C
C      END
C

```

Simple DO-loop

In order to count the data set number, the initialization of the counter, its increment, a test, and transferring the control to the statement after counter initialization are all necessary, making the process very cumbersome. All these steps can be summarized to a simple **DO** loop, which has the following syntax:

```

      DO n {counter variable} = n1, n2, nstep
          statement(s)
n      CONTINUE

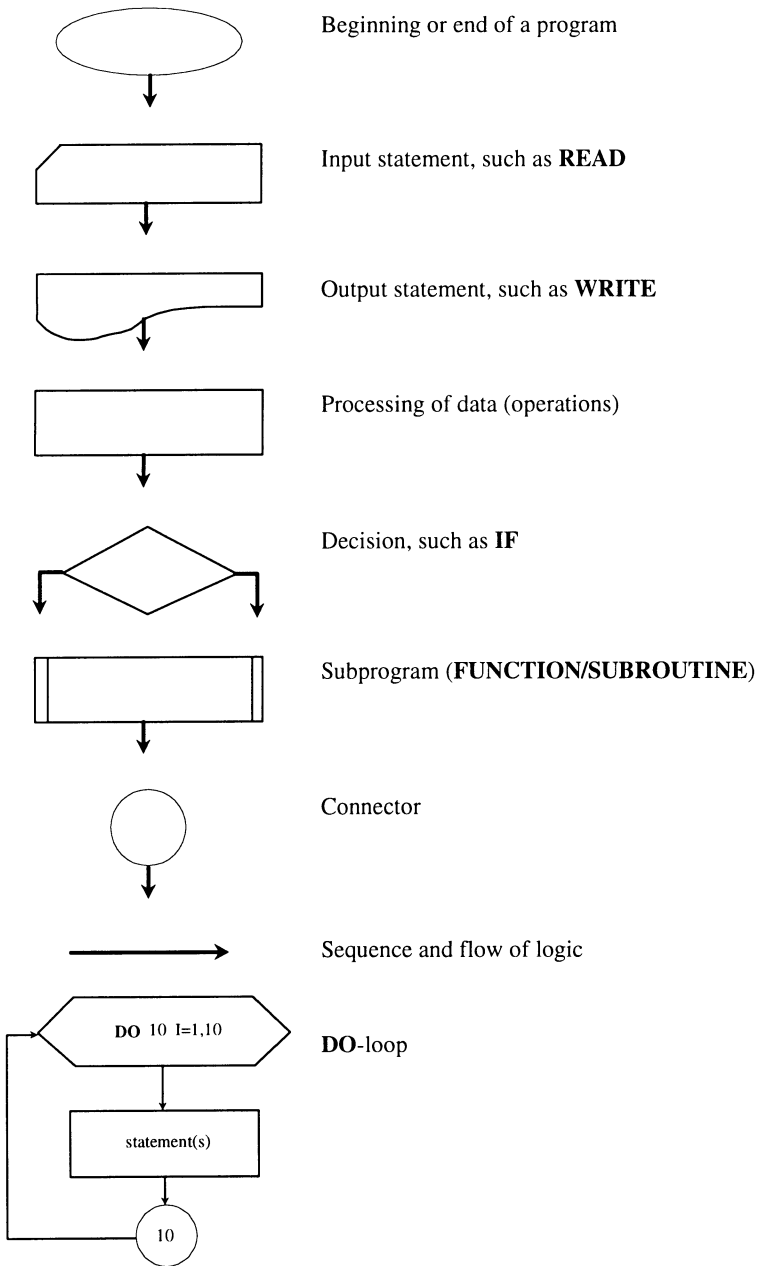
```

where,

n : statement number (an integer value, e.g., 10)
 counter variable : starts with I, J, K, L, M, or N
 n₁, n₂ : initial and final values of counter (integers), respectively,
 nstep : increment size of counter, default is 1.

Flow symbols/chart/diagram

It will be easier to explain the function of **DO** loop with the help of flow symbols. Let us learn each of these symbols to represent various functions of a program. Such symbolic representation of a program is very helpful in the process of planning and documenting the same to solve a specific problem. Also such a representation, known as a flow chart or a flow diagram, makes it easy to understand what an otherwise complicated program is actually doing. The following are the most common symbols used:



Let us examine the flow diagram of a **DO**-loop. When the compiler sees the **DO**-loop for the first time, the counter **I** assumes the first value, i.e., 1 and

checks if it is greater than the last value (= 10). Until this condition is satisfied, it continues in the loop, and, at the statement number 10 (**CONTINUE**), the counter is automatically incremented by the counter step (if no step is specified it is incremented by the default value 1) and goes back to the initial statement of the **DO**-loop. When **I** becomes greater than 10, the control is transferred out of the loop. After normal completion of the **DO**-loop, the counter assumes a value equal to its last count plus the step. So, out of the loop, **I** will have a value of **11** in this case. The counter can also progress in the opposite direction (the initial value of counter is greater than its terminal value), that is, with a negative step value, **I** may be reduced at each loop by the step size. When **I**, in this case, becomes less than the last count, the loop ceases to execute.

If the step size is denoted by **K**, which may be a positive or a negative integer constant, and the last count as **N**, the value of the counter outside the loop after its normal termination will be equal to $(N+K)$. Thus, the direct use of the counter variable outside the **DO**-loop should be made with caution.

Let us rewrite the previous program after adding the **DO**-loop in it.

Example program 2.7

```

C _____
C
C   START A DO-LOOP TO READ DATA SEVERAL TIMES
C
C       DO 10 ICOUNT=1,10
C
C           WRITE(*,*) 'Give values of A, B and C'
C           READ(*,*) A,B,C
C
C           D = B**2 - 4.0*A*C
C
C       TEST IF D IS NEGATIVE OR A IS ZERO. IF NEGATIVE GO BACK
C       TO READ NEW VALUES OF A, B, C
C
C           IF((D .LT. 0.0) .OR. (A .EQ. 0.0)) THEN
C               WRITE(*,*) 'Change values of A, B & C'
C               GOTO 10
C           ELSE
C               R1 = (-B + SQRT(D))/2.0/A
C               R2 = (-B - SQRT(D))/2.0/A
C           ENDIF
C
C           WRITE(*,*) R1,R2
C
C       GO BACK TO INITIAL DO STATEMENT

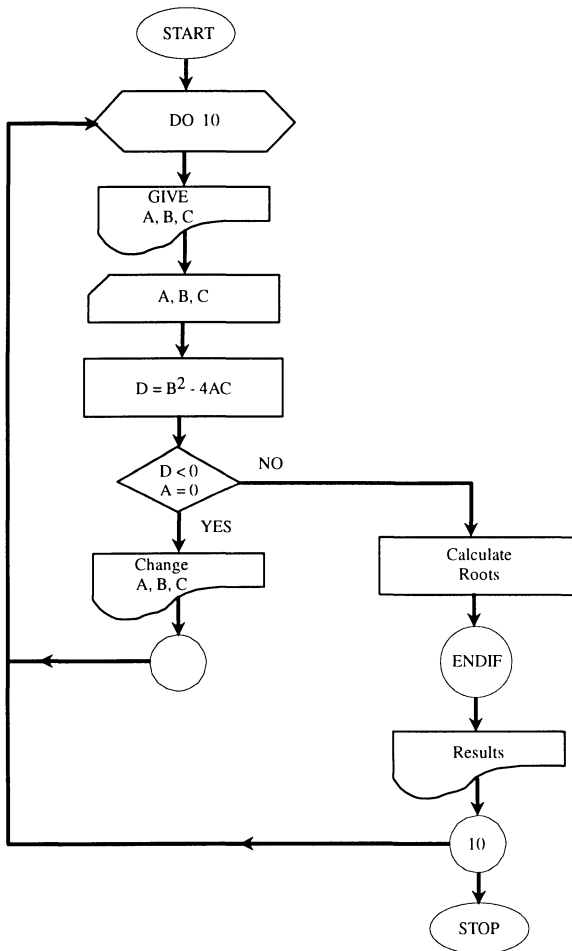
```

```

C
10  CONTINUE
C
      STOP
      END

```

To enhance readability of the program, **DO-loop** and block **IF** are indented as shown in the previous two program listings. This is a very good habit and should always be practiced. Now, let us make a flow diagram of the program to appreciate the advantage of it in understanding various features of the same.



Arrays — subscripted variables

For a moment let us divert our attention to another type of problem, which involves a large group of numbers. Since, in programming, numbers are denoted by appropriate variable names, a large group of numbers will require equal number of variables, which may be inconvenient to handle efficiently. Instead, if subscripted variables are used, the problem turns out to be simple. The mathematical concept of subscripts to identify data items can be directly used in FORTRAN.

A one-dimensional array can identify a quantity or a name with respect to its position. A two-dimensional array, in the same token, can identify a quantity by its two properties and a three-dimensional array by its three properties. In case of a single-dimensional array, there is only one subscript, two and three subscripts in case of two- and three-dimensional arrays, respectively. Mathematical notation uses lowered/subscripted number(s), but, in case of FORTRAN, the subscripted number(s) is/are used under parentheses after the name of the quantity.

If a person is defined by the variable name X, and he or she is identified by age and height classes, I and J, respectively, where I = 1, will group the young age ranging from 10 to 15 years, I = 2 for ages 16 to 20, I = 3 for ages 20 to 40, and I = 4 for ages above 40 years, and J = 1 for a height up to 1 m, J = 2 for a height range of 1.1 to 1.5 m, and J = 3 for height above 1.5 m. Thus the subscripted variable X(3,2) will identify persons having an age range between 20 and 40 years and their range of heights between 1.1 and 1.5 m. This is an example of a two-dimensional array.

A person can also be identified by age (I), height (J), and weight (K). In this case it will be a three-dimensional array, X(I,J,K). The subscripts in FORTRAN are always integer numbers, and, by convention, the row is always written first followed by the column. All items in an array are of the same data type, which may be integer or real, and the first letter convention identifies the type of the array. A subscripted array is called a **matrix**.

DIMENSION statement

For a simple variable name, the compiler assigns a memory location. In the case of a subscripted variable, there should be more than one memory location and the programmer should specify the maximum size of the subscript(s) to the compiler. The statement for doing this is **DIMENSION**, which reserves required memory locations for the particular array irrespective of how much of it is really being used during the program execution. This statement is generally placed in the beginning of a program before the first use of the subscripted variable. The basic form of **DIMENSION** is given below:

DIMENSION X(n), Y(n₁,n₂), Z(n₁,n₂,n₃), M(n₁,n₂, ... , n_k)

where,

X, Y, Z, M : array names

n₁, n₂, n₃, ... , n_k : subscripts (integer constants)

The subscripts in a **DIMENSION** statement should not be variable names unless they are previously declared by **PARAMETER** statement(s). They should explicitly be integer constants and be used with discretion, meaning that an array should not be overdimensioned.

PARAMETER statement

The **PARAMETER** statement should be the first statement of a program followed by other declaration statements, such as **DIMENSION**, **CHARACTER**, etc. It is declared as follows:

PARAMETER (I=50,J=50,IMAX=100,JMAX=100)
DIMENSION X(I,J),Y(IMAX,J),Z(IMAX,JMAX)

The constant names, such as I, J, IMAX, and JMAX can be used in any statement to refer to the constant. Thus the statement **IF(A .EQ. I) STOP** is valid, but the values of the names cannot be altered by any statement in the program. Any **DO**-loop in the program should not use I, J, IMAX, or JMAX as its counter, which will let the values of these names alter.

If two 4 x 6 matrices called A and B are to be added to form matrix C, the following program segment is written to do this:

```

C
C
C      ADDITION OF TWO 2-DIMENSIONAL ARRAYS
C
C          DIMENSION A(4,6),B(4,6),C(4,6)
C
C          DO 20 I=1,4
C              DO 10 J=1,6
C                  C(I,J) = A(I,J) + B(I,J)
10              CONTINUE
20          CONTINUE

```

Each of the matrices in the above example has 24 (4x6) elements. The first time through with $I = 1$, the inner **DO** 10 loop will be completed to calculate the first row elements of C matrix. Then with $I=2$, the second row elements until the fourth and the last row elements are calculated, the **DO**-loop will continue. Thus C matrix also will have 24 elements in it.

In the above example, the **DIMENSION** statement could be preceded by a **PARAMETER** statement defining the array size,

```
PARAMETER (IX=4,JX=6)
DIMENSION A(IX,JX),B(IX,JX),C(IX,JX)
```

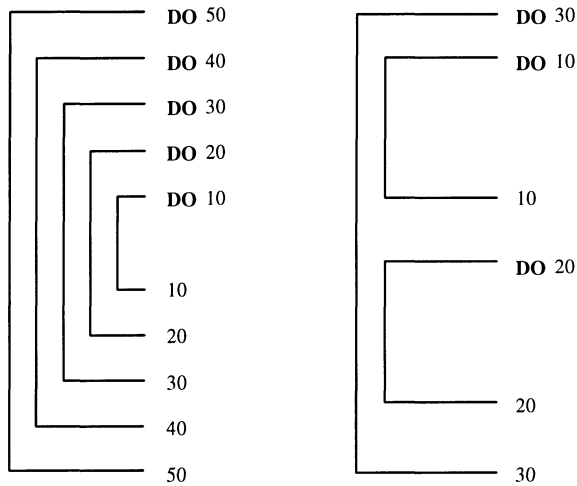
but the constant names IX and JX should not be used as counters substituting I and J in the two **DO**-loops, or, in other words, counters I and J should not be used as constants declared in the argument list of **PARAMETER**.

This is the proper place to explain the rules for **DO**-loops. This feature is very strong in FORTRAN and is extremely useful. The rules are as follows:

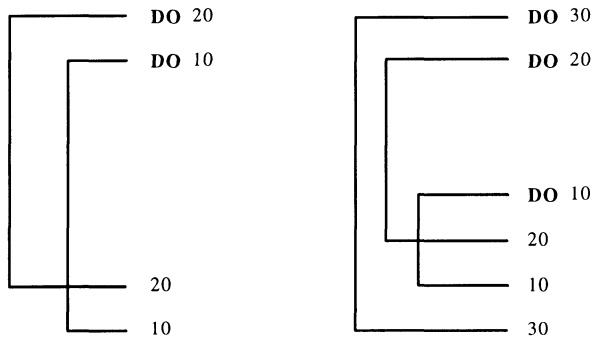
1. The parameters of a **DO** statement defined once must not be altered inside the loop. This includes the counter, its initial and final counts, and the increment.
2. The value of the counter outside the loop is not equal to its final count number. Thus its use outside the loop after a normal exit should be made with caution.
3. **DO**-loops should be entered through only the first **DO** statement. From the outside of a **DO**-loop the control should never be transferred to its inside range. The contrary, that is, exiting a **DO**-loop before its normal termination, is permitted.
4. If there is more than one **DO**-loop, the inner one should be completely inside the next outer one and so on. The diagrammatic representation below will make it clear:

In the first legitimate example (see diagram next page), five **DO**-loops can have the same statement number and thus the same terminal without altering their combined function. Nevertheless, it is advised to practice as shown above for better clarity.

In regression analysis and many other calculations, the technique of summing a group of numbers is of utmost importance and it will be good practice to learn this technique now and, at the same time, apply our knowledge of array, subscripted variable, **DO**-loop, etc.



The following schematic representation explains the situations that should not be allowed to occur.



Nested **DO**-loop not permitted

If there are n numbers, and it is desired to calculate the standard deviation among them, the expression for the standard deviation, S , should be known or be given. It is as follows:

$$S = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{n - 1} \quad 2.3$$

where,

X_i are the data points, and

$\bar{X}$: Arithmetic mean of n data points

$$\bar{X} = \frac{\sum_{i=1}^n X_i}{n} \quad 2.4$$

External input and output files

Here it will then be necessary to compute sums of two series, which may be denoted by two variables XMEAN ($\bar{X}$ in Eq. 2.4) and STDEV (S in Eq. 2.3). Each should be initialized to zero outside the loops for summing up. When there are a large number of data points, it is convenient to read their values from an external file rather than providing them from the keyboard during execution of the program. For input data there should be an existing external file containing these data. If its name is DATA.IN, the following statement is used in order to open this file:

OPEN(n,FILE='file name',STATUS='OLD')

where, **n** is the unit number to specify the file. The file name must be according to FORTRAN grammar, which applies equally to any legal variable name in FORTRAN, the only difference being in the case of a file name there may also be an extension name. The extension name starts after the principal file name with a period in-between. It is meant to help identify the function of a file, e.g., DATA.IN indicates that it is an input data file. Similarly, DATA.OUT indicates that it is an output file, DRUM.TXT for a text file etc. Unlike the principal name, it can start with a number, and the string can have a maximum of three characters, which may be only letters, or only characters, or any combinations of them.

In case of an output file, it may already exist (**STATUS= 'OLD'**), or a new file can be created (**STATUS='NEW'**) to write on it.

OPEN(n,FILE='file name',STATUS='NEW')

Each time a file with **STATUS='NEW'** is opened by the above **OPEN** statement, it loses its previous contents. The access to such external files is sequential as a default action when nothing is specified about it.

A file thus opened can be closed with the following **CLOSE** statement:

CLOSE(n,STATUS='KEEP')

The program to calculate standard deviation is given below, which later will be added as another feature to our previous program of computing real roots of a quadratic equation.

Example program 2.8

```
C
C
C PROGRAM TO CALCULATE STANDARD DEVIATION OF N DATA POINTS
C
C      DIMENSION X(100)
C
C INPUT AND OUTPUT EXTERNAL FILES ARE OPENED
C
C      OPEN(10,FILE='DATA.IN',STATUS='OLD')
C      OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C XMEAN AND STDEV ARE INITIALIZED TO ZERO
C
C      XMEAN = 0.0
C      STDEV = 0.0
C
C NUMBER OF DATA POINTS AND DATA ARE READ - XMEAN IS COMPUTED
C
C      WRITE(*,*) 'Give the number of DATA points and <ENTER>.'
C      READ(*,*) N
C
C      DO 10 I=1,N
C          READ(10,*) X(I)
C          XMEAN = XMEAN + X(I)/N
10  CONTINUE
C
C NOW THE LOOP FOR CALCULATION OF STANDARD DEVIATION
C
C      DO 20 I=1,N
C          STDEV = STDEV + (X(I) - XMEAN)**2/(N - 1)
20  CONTINUE
C
C      STDEV = SQRT(STDEV)
C
C      WRITE(*,*) 'Standard deviation = ',STDEV
C
C WRITE THE RESULT ALSO IN THE OUTPUT FILE 'DATA.OUT'
C
C      WRITE(20,*) 'Standard deviation = ',STDEV
C
C CLOSE BOTH EXTERNAL FILES
C
C      CLOSE(10,STATUS='KEEP')
```



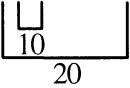
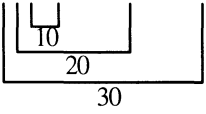
```

CLOSE(20,STATUS='KEEP')
C
      STOP
      END
C_____

```

Implied DO-loops in READ and WRITE

When input data are read in free format inside a **DO**-loop as in the case of the program above, data should be provided vertically, i.e., each datum on a separate line in the file. This may not be very convenient when there are many data points. An implied **DO**-loop is a very convenient way to express one or more than one **DO**-loops in one line. An implied **DO**-loop in **READ** statement will allow entering data horizontally with a comma or a space between them. The syntax of the implied **DO**-loop in **READ** and **WRITE** statements is identical and is as follows:

	<u>Regular DO-loop</u>	<u>Equivalent Implied DO-loop</u>
	<pre> DO 10 I=1,L READ(10,*) X(I) 10 CONTINUE </pre>	<pre> READ(10,*)(X(I),I=1,L) </pre> 
	<pre> DO 20 I=1,L DO 10 J=1,M READ(10,*) X(I,J) 10 CONTINUE 20 CONTINUE </pre>	<pre> READ(10,*)((X(I,J),J=1,M),I=1,L) </pre> 
	<pre> DO 30 I=1,L DO 20 J=1,M DO 10 K=1,N READ(10,*) X(I,J,K) 10 CONTINUE 20 CONTINUE 30 CONTINUE </pre>	<pre> READ(10,*)((X(I,J,K),K=1,N),J=1,M),I=1,L) </pre> 

Checking for end of data by READ statement

If there is more than one group of data to be read, the **READ** statement can have an **END** specification in its argument list. The form of it is **END = n₃**, where n₃ is the statement number to which control would go when all the data have been read. Thus, the **READ** statement below will be executed until

there are no more data to be read and at that time control will go to statement number 10.

READ(10,*,END=10) (X(I),I=1,N)

This feature is especially useful when the data set number is unknown or is variable. It works only to read data from an external file, but does not work for list-directed input. The program last written can be modified with this statement, and, when it is done, it will not be necessary to supply the number of the data set. When there are no data to read, the **END = 40** in the **READ** statement will transfer the control to the statement number 40 to close both input and output files before stopping. The following modified version of the previous program rewritten in this light will clarify the use of this special feature. Here the maximum number of data points can not exceed 100 as per the **DIMENSION** statement.

Example program 2.9

```

C _____
C
C PROGRAM TO CALCULATE STANDARD DEVIATION OF N DATA POINTS
C
C DIMENSION X(100)
C
C INPUT AND OUTPUT EXTERNAL FILES ARE OPENED
C
C OPEN(10,FILE='DATA.IN',STATUS='OLD')
C OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C XMEAN AND STDEV ARE INITIALIZED TO ZERO
C
C XMEAN = 0.0
C STDEV = 0.0
C
C DATA ARE READ; XMEAN IS COMPUTED
C
10 READ(10,*,END=40) N,(X(I),I=1,N)
C DO 20 I=1,N
C XMEAN = XMEAN + X(I)/N
20 CONTINUE
C
C NOW THE LOOP FOR CALCULATION OF STANDARD DEVIATION
C
C DO 30 I=1,N

```

```

          STDEV = STDEV + (X(I) - XMEAN)**2/(N - 1)
30  CONTINUE
C
          STDEV = SQRT(STDEV)
C
          WRITE(*,*) 'Standard deviation = ',STDEV
C
C  WRITE THE RESULT ALSO IN THE OUTPUT FILE 'DATA.OUT'
C
          WRITE(20,*) 'Standard deviation = ',STDEV
C
C  GO TO READ NEW DATA
C
          GOTO 10
C
C  CLOSE BOTH EXTERNAL FILES
C
40  CLOSE(10,STATUS='KEEP')
    CLOSE(20,STATUS='KEEP')
C
    STOP
    END
C_____

```

FORMAT-directed input and CHARACTER statement

So far, free- or unformatted **READ** and **WRITE** statements have been used in all the previous programs. The free formatted **READ** statement is very easy to use and will be mostly used in almost any situation except when it will be required to read character data. Reading character data will require a **FORMAT** statement. Let us demonstrate this through one of the previous example programs 2.3, where an endless loop was terminated by testing a given unique value of one of the input variables. A more elegant way to stop or continue a program is to use a character string S, which can have a single character value Y or N where Y will mean **Y**es to continue and N to stop or **N**ot to continue. In order to declare S as a character variable that can bear one character value, the **CHARACTER** statement should be placed in the beginning of the program having the following syntax:

CHARACTER *1 S

Its general form is

CHARACTER *n S1,S2,S3, ...

where each of the character variables S1, S2, S3, etc. can bear a maximum of n characters.

Example program 2.10

```

C _____
C
      CHARACTER *1 S
C
10  WRITE(*,*) 'Press Y to CONTINUE; N to STOP'
      READ(*,20) S
      IF(S .EQ. 'N') STOP
C
      WRITE(*,*) 'Give values of A, B and C'

      READ(*,*) A,B,C
C
      D = B**2 - 4.0*A*C
C
      R1 = (-B + SQRT(D))/2.0/A
      R2 = (-B - SQRT(D))/2.0/A
C
      WRITE(*,*) R1,R2 C
C  GO BACK TO READ MORE DATA ON A, B AND C
C
      GOTO 10
C
C  FORMAT STATEMENT
C
20  FORMAT(A1)
C
      STOP
      END
C _____

```

The **READ** statement in the above example has a number as its second argument instead of the asterisk that specifies the number/label of the **FORMAT** statement. To write the values of character variables, the letter A is used in the format statement with an integer constant by its right hand side. It is in the following form:

m **FORMAT**(An)

where,

m : statement number (between first and fifth column)

An : a character variable field containing n characters including spaces or blanks.

FORMAT-directed output

Unlike in **READ** statements where formatting is used only in the case of character variables, formatted output is almost always preferred. The format directed input and output require a pair of statements, the **READ/WRITE** statement and the **FORMAT** statement. The latter specifies the form of the data being read in or written out. One **FORMAT** statement can be used by more than one input or output statements. It is not necessary that a **READ** or **WRITE** statement should be accompanied by its **FORMAT** statement. In fact, it is a good practice to group all the format statements in one place just before the **STOP** statement.

The **FORMAT** statement consists of a statement number followed by the keyword **FORMAT**. The sets of specifications are put after the word **FORMAT** enclosed in parentheses. Each specification set consists of three elements — (i) a single-letter edit descriptor, (ii) field size, and (iii) decimal location.

The one letter edit descriptor can specify real, integer, real exponent, or character data by the letters F, I, E, and A, respectively. The field size is the total number of positions the data will occupy including the sign + or - and the decimal point, if any, and decimal location tells about the number of positions to the right of the decimal. The decimal location is only used in the case of real data.

The specification set **Fw.d** defines editing of real data having a field size of **w** positions and **d** positions to the right of the decimal point. Thus if a real variable has a value 125.87676 the **FORMAT** statement for **WRITE** with **F8.4** specification set, will write it as 125.8768. The last digit on the right hand side will be rounded off. If the number has more than three digits before the decimal point, such as 1255.2536, the aforesaid format specification will not be able to handle it, and the number will be replaced by eight asterisks. The edit descriptors **I** for integer variables and **A** for character variables have the forms as **Iw** and **Aw**, respectively, where **w** is the field size. The number on output is always right-justified.

When the calculated value of a real variable is unpredictable, meaning that its value varies from a very small fraction to a very large number, the **F** specification cannot be used. In this case, the edit descriptor **E** in its exponent form is extremely convenient. The specification set **Ew.d** tells that the field size is **w** including the sign and the exponent **E+mn**, where **m** and **n** can

assume any value between 0 and 9, and **d** is the number of digits to the right of the decimal point. An **E** specification does not allow any digit on the left hand side of the decimal point, except zero. Thus it requires at least seven digit positions more than the size of the fraction to allow one space for the sign, one space for zero, one space for the decimal point and four spaces for the exponent **E+mn**.

In some implementation of FORTRAN, the zero does not exist before the decimal point, and hence, it needs at least six spaces more than the number of digits in the fraction. The numbers -6574.8765 and 16574.8765 with **E15.8** will be expressed as follows:

$$\begin{aligned}-6574.8765 &\Rightarrow -0.65748765\text{E}+04 \\ 16574.8765 &\Rightarrow 0.16574877\text{E}+05\end{aligned}$$

If the number has more digits than the digits in the fraction, **d**, its **E** representation will truncate the quantity after rounding off. If it is desired to keep **k** digits before the decimal point, its syntax is **kPEw.d**, where **w** should be at least equal to or greater than **d+6+k**. Thus the above numbers with the format specifier **1PE15.8** will be as follows:

$$\begin{aligned}-6574.8765 &\Rightarrow -6.57487650\text{E}+03 \\ 16574.8765 &\Rightarrow 1.65748765\text{E}+04\end{aligned}$$

When more than one variable can be expressed by the same format specifier, an integer constant before this specifier can make it repeat by that integer number. Thus **4F10.6** will make the specifier **F10.6** repeat four times.

The **FORMAT** statement follows the **WRITE** statement, and it has a label number specified as the second argument of **WRITE**. There are some edit specifiers that are used with **FORMAT** statements for horizontal and vertical spacing between one or more data values including characters.

For horizontal spacing, the following edit specifiers are used:

nX : n spaces blanks to the right
Tn : Tabulate starting from column n

For vertical spacing, the slash (/) is used. One slash after a statement will cause the current line to terminate, and whatever will be put after the slash will appear on the next line. Thus **n** number of slashes will cause **n** lines to be skipped creating **n** blank lines. This feature is also useful to clear the screen when necessary. The first position inside parentheses of **FORMAT** statement for **WRITE** is used for carriage control or vertical spacing. Also the following single characters enclosed by single apostrophes and placed in the first position will control the vertical positioning:

'0' : skipping two lines
 '1' : skip to first line of next page
 '+' : no advance

The "no advance" feature is useful when overwriting is done with two identical formatted **WRITE** statements in order to print bold face. Between the two identical write statements, the second should have '+' in its **FORMAT** statement. Here it should be noted that some printing terminals do not use a carriage control character. Thus the character put in the first position is displayed or printed without showing its effect as described.

The characters enclosed by single apostrophes in the **FORMAT** statement will appear on the screen or will be printed by the printer exactly how they have been entered minus the apostrophes.

The features explained above will be clear through some examples:

```
WRITE(*,10) X,I
10 FORMAT(1X,F10.6/2X,'Value of Counter = ',I2)
```

The above **FORMAT** with label **10** has its first position blank with **1X**, followed by **F10.6** for the variable **X**. Then the line breaks by the slash and on the next line, allowing first two spaces blank using **2X**, the value of the variable **I** is written by **I2** identifying it as "Value of Counter" with apostrophe edit descriptor. In case of a printing terminal using a carriage control character, the first blank position is required, and, if it is not provided, the first character or digit will be lost at the time of printing.

The following two statements will yield the same results :

```
10 FORMAT('0',F10.6/2X,'Value of Counter = ',I2)
and
10 FORMAT(//,F10.6/2X,'Value of Counter = ',I2)
```

Here the comma after '0' and '/' is optional. There are two ways to print bold face,

```
(i) WRITE(*,10)
10 FORMAT(T10,'Bold face',T10,'Bold face')

(ii) WRITE(*,10)
10 FORMAT(9X,'Bold face')

WRITE(*,20)
20 FORMAT('+',9X,'Bold face')
```

Repetition of FORMAT specifications

The repetition of format specifications is explained with the given **FORMAT** statements for **WRITE**:

FORMAT(1X,E15.8,2X,410.6) : One blank, E-format specification, two blanks and repeat F10.6 four times

FORMAT(1X,5(F6.2,2X,I2)) : One blank, repeat (F6.2,2X,I2) five times

FORMAT(20(/)70('*')) : Twenty blank lines, usually for clearing screen, followed by a line made by 70 asterisks.

In FORTRAN it is very convenient to print output data with the minimum use of format specifiers. If the number of variables in the list of **WRITE** is not equal to the number of specifiers in the **FORMAT** statement, the output still can be written without an error. In the case where there are more variables than the number of format specifiers, the format specification(s) is/are repeated at the most recent left parentheses. When there are more specifications than the number of variables, extra specifications on the right are ignored.

```
WRITE(*,10) (X(I),Y(I),Z(I),I=1,10)
10 FORMAT(1X,3(1PE15.7,2X))
```

Here, the one value each of X, Y, and Z will be written on the first line using **1PE15.7** followed by two spaces, next values of X, Y, and Z on the second line, and so on. Thus there will be a total of ten lines of output.

```
WRITE(*,10) A,B,C
10 FORMAT(1X,5(F10.2,2X))
```

In the case above, values of A, B, and C are written with **F10.2** allowing two blanks between any two values, and the two remaining specifications are ignored.

Now, let us incorporate some of the features we just learned in our Example 2.7.

Example program 2.11

```

C
C
C   START A DO-LOOP TO READ DATA SEVERAL TIMES
C
C       WRITE(*,20)
C       READ(*,*) N
C
C       DO 10 ICOUNT=1,N
C
C           WRITE(*,30)
C           READ(*,*) A,B,C
C
C           D = B**2 - 4.0*A*C
C
C           TEST IF D IS NEGATIVE OR A IS ZERO. IF NEGATIVE GO BACK
C           TO READ NEW VALUES OF A, B, C
C
C               IF(D .LT. 0.0) THEN
C                   WRITE(*,40)
C                   GOTO 10
C               ENDIF
C               IF(A .EQ. 0.0) THEN
C                   WRITE(*,50)
C                   GOTO 10
C               ENDIF
C               R1 = (-B + SQRT(D))/2.0/A
C               R2 = (-B - SQRT(D))/2.0/A
C               WRITE(*,60) R1,R2
C
C           GO BACK TO INITIAL DO STATEMENT
C
C       10   CONTINUE
C
C
C   *****  FORMAT  STATEMENTS  *****
C
C       20   FORMAT(/10X,'Give the number of quadratic equations'/)
C       30   FORMAT(2X,'Give values of A, B and C'/)
C       40   FORMAT(2X,'Root of a negative number. Give NEXT A, B & C'/)
C       50   FORMAT(2X,'A can not be ZERO. Give data for the next set'/)
C       60   FORMAT(2X,'The value of the FIRST root = ',1PE15.7/2X,
C       1           'The value of the SECOND root = ',1PE15.7/)

```

C

STOP

END

C

Example 2.11 has been further modified mainly with respect to detailed formatted outputs. Also the number of data points are initially read, thus offering more flexibility of reading exactly the desired number of data. It is interesting to note that the **FORMAT** statements are not placed immediately after corresponding **WRITE** statements. Instead, these have been grouped in one place just before the statement **STOP**. This makes the program easier to read and brings clarity and elegance to it.

Branched block IF

The use of two block-**IF** statements in the above example allows us to place two **WRITE**s in order to advise the user about the hazard ahead. These two blocks deviate from the standard syntax of block-**IF** explained earlier in the sense that the keyword **ELSE** in either case has been omitted. However, at this point it should be clarified that this form is also legal. The block-**IF** has another structure that facilitates joining more than one block-**IF** just explained with the help of **ELSEIF ... THEN** in place of **ELSE** and without using a separate **ENDIF**:

```

IF (condition) THEN
    statement(s)
ELSEIF (condition) THEN
    statement(s)
ELSEIF (condition) THEN
    statement(s)
.....
ELSE
    statement(s)
ENDIF

```

Thus the two block-**IF**s used in example program 2.11, can be replaced by the following branched block-**IF**:

```

IF(D .LT. 0.0) THEN
    WRITE(*,40)
    GOTO 10
ELSEIF(A .EQ. 0.0) THEN
    WRITE(*,50)

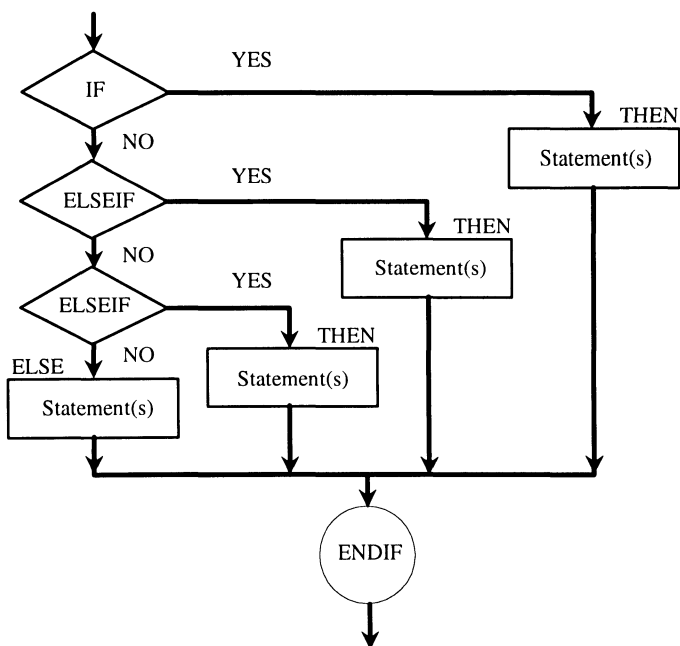
```

```
GOTO 10
ENDIF
```

In this case also the keyword **ELSE** was not necessary, hence omitted; nevertheless, it can be added with a **CONTINUE** statement:

```
IF(D .LT. 0.0) THEN
  WRITE(*,40)
  GOTO 10
ELSEIF(A .EQ. 0.0) THEN
  WRITE(*,50)
  GOTO 10
ELSE
  CONTINUE
ENDIF
```

The flow diagram of such a block-**IF** is given as follows:



Just like **DO**-loops, more than one block-**IF** statements can be nested one inside the other(s) and the rules 3 and 4 for nested **DO** loops, explained earlier, do apply.

There are two intrinsic functions to convert data from one type to another, and they are:

FLOAT (integer number or expression)	: Convert to real type
IFIX (real number or expression)	: Convert to integer truncating fractional part.

Intrinsic/library functions

There are some intrinsic/library functions in FORTRAN that are very handy when a specific need arises. These functions are listed in Table 2.1

REAL, INTEGER, and DOUBLE PRECISION statements

The default action of FORTRAN of treating a variable name with any of the six letters I, J, K, L, M, and N or starting with any of these letters as the integer type can be overruled by the following two declaration type statements that should be placed at the beginning of the program along with **DIMENSION** and **CHARACTER**:

```
REAL J,ICOUNT,KX
INTEGER RESULT,A,B,C
```

The double precision accuracy can be attributed to one or more variables by the following statement, which also should be placed in the beginning along with declaration statements **REAL, INTEGER** etc.:

```
DOUBLE PRECISION X,Y,Z
```

In Table 2.1, a list of double precision library functions is presented.

DO WHILE and EXIT statements

It has been observed so far that in order to go out of a **DO**-loop before its normal termination, which may occur if a condition inside the loop is satisfied earlier, a **GOTO** should be the answer. A conditional **DO WHILE** is an improvement on this limitation of the usual **DO**-loop.

Table 2.1. FORTRAN Library Functions.

Definition	Type of Argument and Function		
	Real	Integer	Double-precision
Absolute value	ABS	IABS	DABS
Remainder of a division	AMOD	MOD	DMOD
Largest value	AMAX1	MAX0	DMAX1
Smallest value	AMIN1	MIN0	DMIN1
Natural logarithm (base e)	ALOG	ALOG	DLOG
Common logarithm (base 10)	ALOG10	ALOG10	DLOG10
Exponential	EXP	EXP	DEXP
Square root	SQRT	SQRT	DSQRT
Sine	SIN	SIN	DSIN
Cosine	COS	COS	DCOS
Tangent	TAN	TAN	DTAN
Arcsine	ASIN	ASIN	DASIN
Arccosine	ACOS	ACOS	DACOS
Arctangent (one argument)	ATAN	ATAN	DATAN
Arctangent (two arguments)	ATAN2	ATAN2	DATAN2
Hyperbolic sine	SINH	SINH	DSINH
Hyperbolic cosine	COSH	COSH	DCOSH
Hyperbolic tangent	TANH	TANH	DTANH

Its syntax is as follows:

```

DO WHILE (condition(s))
    statement(s)
END DO

```

There may be one or more logical conditions, which, until they are satisfied, the loop continues. In some versions of FORTRAN compiler, it

ends with an **END WHILE** instead of an **END DO**. Another recent version of the regular **DO**-loop eliminates the label and the **CONTINUE** statement as shown below:

DO I=1,10		DO 10 I=1,10
statement(s)	replaces	statement(s)
END DO		10 CONTINUE

There is another more elegant and structural way to go out of a **DO**-loop than using a **GOTO**. It is done by the keyword **EXIT** following a logical condition. It transfers control from within a **DO**-loop to the first executable statement following the end of the loop. We will see its use and the use of others in the following example program:

Example program 2.12

```

C _____
C
C PROGRAM TO CALCULATE STANDARD DEVIATION OF ONE OR MORE SET OF DATA
C POINTS SAVED IN AN INPUT FILE.
C
C      DIMENSION X(100)
C      REAL MEAN
C      INTEGER COUNT
C      CHARACTER *1 S
C
C INPUT AND OUTPUT EXTERNAL FILES ARE OPENED
C
C      OPEN(10,FILE='DATA.IN',STATUS='OLD')
C      OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C DATA ARE READ; NUMBER OF DATA POINTS CAN NOT EXCEED 100
C VARIABLES MEAN AND STDEV ARE INITIALIZED TO ZERO
C
C      MEAN = 0.0
C      STDEV = 0.0
10  READ(10,*,END=40) COUNT,(X(I),I=1,COUNT)
C      DO 20 I=1,COUNT
C          MEAN = MEAN + X(I)/COUNT
20  CONTINUE
C
C NOW THE LOOP FOR CALCULATION OF STANDARD DEVIATION
C

```

```

DO 30 I=1,COUNT
    STDEV = STDEV + (X(I) - MEAN)**2/(N - COUNT)
30  CONTINUE
C
    STDEV = SQRT(STDEV)
C
    WRITE(*,50) COUNT,STDEV
C
C  WRITE THE RESULT ALSO IN THE OUTPUT FILE 'DATA.OUT'
C
    WRITE(20,50) COUNT,STDEV
C
C  IF THERE ARE MORE DATA GO BACK TO READ THEM, OTHERWISE STOP
C
40  WRITE(*,60)
    READ(*,70) S
    IF((S .EQ. 'N') .OR. (S .EQ. 'n')) STOP
    GOTO 10
C
C  CLOSE BOTH EXTERNAL FILES
C
    CLOSE(10,STATUS='KEEP')
    CLOSE(20,STATUS='KEEP')
C
C  ***** F O R M A T   statements *****
C
50  FORMAT(2X,'Standard Deviation among ',I3,' Data points ',
1      'is ',F10.6//)
60  FORMAT(2X,'If have MORE data, Press [Y], ELSE [N] and',
1      '<ENTER>')
70  FORMAT(A1)
C
    STOP
    END
C

```

The above program shows the use of intrinsic declaration functions **REAL** and **INTEGER** and another new feature of testing two conditions inside the same **IF** statement. In the example program 2.9, it was our experience that, when it was run, the pressing of the lower case letter **n** did not stop the program. If we go back to the initial part of this part, where the binary digits and bytes were discussed, we understand that the internal representation of upper case **N** and that of lower case **n** are not the same. Thus in the above example program, both the upper and lower case letters

were tested by the **IF** statement so that pressing either of them will stop the program.

Arithmetic IF and computed GOTO statements

In older FORTRAN programs the following two statements were frequently used. These are arithmetic **IF** and computed **GOTO** statements. The block-**IF** structure can very efficiently replace both these statements, but in order to help readers understand older FORTRAN programs these are explained below:

IF(arithmetic expression) n_1, n_2, n_3

where, the arithmetic expression, when evaluated, is negative, control goes to the first statement with label n_1 ; if zero, to the second statement with label n_2 and if greater than zero (positive), to the third statement with label n_3 . The following example will explain it better:

IF(X - 10.0) 10,20,30 : Control goes to 10 when $X < 10.0$
 : Control goes to 20 when $X = 10.0$
 : Control goes to 30 when $X > 10.0$

The structure of computed **GOTO** is as follows:

GOTO($n_1, n_2, n_3, \dots, n_k$), I

where the above **GOTO** transfers control to the statement label n_1 for $I = 1$, to label n_2 for $I = 2$, to label n_3 for $I = 3$, ... to label n_k for $I = k$.

DATA statement

As it has been demonstrated by example program 2.12, two variables **MEAN** and **STDEV** are set to an initial value zero. It frequently occurs in statistical calculations. This was done so far by assignment statements before the loops. Another way to do the same is by a **DATA** statement that is placed at the beginning of a program after some declaration specifications, such as **DIMENSION**, **CHARACTER**, etc. but before any executable statements. The **DATA** statement can provide initial values of variables, arrays, and array elements. It has the following syntax:

DATA vname(s) /vvalue(s)/vname(s)/vvalue(s)/

where, vname(s) is the list of variable name(s) and vvalue(s) is the list of value(s) of corresponding variable(s). Its examples are as follows:

DATA XMEAN,STDEV /0.0,0.0/ or **DATA XMEAN,STDEV /2*0.0/**

In the above example, XMEAN and STDEV were initialized as zero.

DATA (X(I),I=1,100) /100*0.0/ICOUNT/1/

This **DATA** statement initializes 100 values of X as zero and gives a unity value to ICOUNT.

DATA X /100*0.0/ICOUNT/1/

It has the same function as the previous example. Thus the advantage of the **DATA** statement becomes obvious. It uses less space, is more efficient, and offers more clarity than doing the same thing by assignment statements.

EQUIVALENCE statement

In the course of writing a lengthy program, it might not be too unusual if the programmer discovered that different variable names were used for the same quantity. Instead of attempting to change all the names in the entire program, the programmer may use a nonexecutable statement **EQUIVALENCE** to exercise control over the assignment of memory locations, thus correcting the error in a much simpler way. If the variables I, I1 and J1 were used to represent the same value in the program, the following statement will force them to share the same memory location:

EQUIVALENCE (I,I1,J1)

An equivalence can also be set up among more than one group of names, such as

EQUIVALENCE (I,I1,J1),(X,X1,X2),(A,B)

This statement can be used to conserve memory space. If X, Y, Z represent large two-dimensional arrays within a program, then only one of these arrays is needed at any given time, or in other words, there is no overlapping among these three variables. By an **EQUIVALENCE** statement one-third memory space can be saved in this particular case:

```
DIMENSION X(50,10),Y(50,10),Z(50,10)
EQUIVALENCE (X(1),Y(1),Z(1))
```

Though it may appear that only one value of each variable is being equated, actually the effect is much deeper. Arrays are always stored in successive memory locations by the compiler. The external representation of an array is done by the concept of the location with respect to row and column. For example, in a 10x10 B matrix, B(1,2) will be the external representation of an element, meaning that it is the element located on the first row and second column. Its internal representation, however, is different. It counts the location number starting by the columns, thus B(1,2) will have the 11-th location in the memory. Aligning the first element of these arrays automatically aligns the rest of the elements. The following example will clarify alignment procedure when subscripted variables appear in an **EQUIVALENCE** statement:

```
DIMENSION X(10),Y(5),Z(5)
EQUIVALENCE (X(5),Y(5),Z(3))
```

```
X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(9) X(10)
Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10)
Z(1) Z(2) Z(3) Z(4) Z(5)
```

Thus aligning the elements in the **EQUIVALENCE** statement denoted by bold letters will automatically align the rest of the elements on their left and right hand sides as shown above. Thus X(1) and Y(1); X(2) and Y(2); X(3), Y(3), and Z(1); X(4), Y(4), and Z(2); X(5), Y(5), and Z(3); X(6), Y(6), and Z(4); X(7), Y(7), and Z(5); X(8) and Y(8); X(9) and Y(9) as well as X(10) and Y(10) will share the same memory locations.

Subprograms — FUNCTIONS and SUBROUTINES

So far what have been written are called **main programs**. It is like an organization where the chief plans all the activities and writes the final report, but he or she gets the work of different sectors done by different subordinates specialized in pertinent activities and who are responsible for their part of the work. The chief is like the main program and his or her subordinates can be compared with the subprograms. If the chief of a large and multifunctional organization tries to do everything by him- or herself, it is going to be a monumental task and it will be impossible to attain the desired efficiency that otherwise could be achieved with proper planning and division of work.

It is exactly what the subprograms are designed for. The purpose of subprograms is to simplify programming and at the same time increase the efficiency of the combined program with respect to (i) ease in decoding/deciphering, (ii) ease in documentation, (iii) reusing program elements, (iv) testing different segments of the program separately, (v) ease in debugging, (vi) ease in designing FORTRAN implementation of an algorithm, and (vii) allowing different programmers to work on different topics of their specialties at the same time.

Subprograms are divided into two categories — (i) **FUNCTION** and (ii) **SUBROUTINE**. In Table 2.1 a list of FORTRAN library functions has been provided. These functions are, in fact, built-in subprograms. These were written and added as a part of the FORTRAN compiler because of their common use in many occasions. It goes without saying that built-in convenience of such subprograms for all possible situations cannot be expected from any compiler. Here comes the use of user-written subprograms, namely, **FUNCTIONs** and **SUBROUTINEs**. A user-oriented subprogram is a complete and independent program that can be invoked or **CALLed** by the main program or by other subprograms. A subprogram receives values from calling program, performs calculations, and then sends back (**RETURNS**) the result(s) to the calling program.

FORTRAN is one of the oldest high level languages and is rich in the user-made software library. This software library consists of user-written subprograms. The advantages of such subprograms besides the efficiency factors just discussed can be listed as follows:

1. These subprograms (multiline **FUNCTION** and **SUBROUTINE**) can be compiled separately, thus avoiding the complexity of compiling an otherwise large program,
2. They split the program into smaller and thus easily manageable parts,
3. These can be made to share the same memory locations for the variables with the main control program, thus relaxing additional memory requirements,
4. Subprograms can be invoked/called by another program or subprogram by a simple program statement, and
5. Common variables in the subprograms can be given the same or different names to those used in the calling programs according to the user's choice, thus making it easier to code and avoid confusion.

Differences between a Program and a Subprogram:

So far the similarities between a program and a subprogram has been discussed. Now let us examine the following differences:

1. In place of a **STOP** statement, the **RETURN** statement is used in a subprogram. This statement transfers the control to the line in the calling program following the invoking/calling statement,
2. **READ** and **WRITE** statements may appear in a subprogram as a special case, but neither is it conventional nor most convenient to provide input or output to a subprogram. The argument list and **COMMON** statement to be discussed later are the efficient tools to this end,
3. A subprogram should have a title for identification as its first statement, which specifies the type, name, and the parameters. The type may be a **FUNCTION** or a **SUBROUTINE**, and the parameters may either be provided in the argument list under parentheses followed by the subprogram title or through **COMMON** statement(s). The title can be a string of a maximum of six letters and/or digits, the first character always being a letter. Unlike the variable name, the first letter convention is not followed in case of subroutines. Thus the subroutine titles starting with integer letters I, J, K, L, M, and N do not make them yield integer values.

The following example of a subprogram placed by the side of a program doing essentially the same task will demonstrate some of the similarities and differences just discussed:

Program	Subprogram
<pre> READ(*,*) A,B,C C C Calculate root C D = B**2 - 4.0*A*C ROOT = (-B + SQRT(D))/2.0/A WRITE(*,*) ROOT STOP END </pre>	<pre> FUNCTION FX(A,B,C) C C Calculate root C D = B**2 - 4.0*A*C FX = (-B + SQRT(D))/2.0/A RETURN END </pre>

Now, a main program can be written that will get the root calculated by a **FUNCTION** subprogram. Thus the following main program in combination with the function subprogram will calculate the real root of a quadratic equation:

```

      READ(*,*) A,B,C
C
C Calculate root by calling the FUNCTION FX
C
      ROOT = FX(A,B,C)

```

```

WRITE(*,*) ROOT
STOP
END

```

It should be noted at this point that the argument list of the **FUNCTION** FX, known as **parameters** of this subprogram, may not necessarily be the same as the variable names used by the main program. Thus the **FUNCTION** subprogram could have been legally written as follows:

```

FUNCTION FX(X,Y,Z)
C
C Calculate the root
C
W = Y**2 - 4.0*X*Z
FX = (-Y + SQRT(W))/2.0/X
RETURN
END

```

Due to their unimportance, the parameters in the **FUNCTION** argument list are called dummy variables. Here the type and order of the variable list in the main- and the subprograms are important and should be preserved.

A **FUNCTION** subprogram has the limitation of returning only one answer at a time to its calling program, though it can be invoked as many times as required. When a subprogram has to compute more than one values, a **SUBROUTINE** subprogram should be used. To demonstrate this, let us calculate both the roots of a quadratic equation:

Function subprogram	Subroutine subprogram
FUNCTION FX(I,X,Y,Z)	SUBROUTINE ROOT(X,Y,Z,R1,R2)
C	C
C I = 1 for the first ROOT	C Both roots are calculated
C I = 2 for the second ROOT	C
C	W = Y**2 - 4.0*X*Z
W = Y**2 - 4.0*X*Z	R1 = (-Y + SQRT(D))/2.0/X
IF(I .EQ. 1) THEN	R2 = (-Y - SQRT(D))/2.0/X
FX = (-Y + SQRT(D))/2.0/X	RETURN
ELSEIF(I .EQ. 2) THEN	END
FX = (-Y - SQRT(D))/2.0/X	
ELSE	
WRITE(*,*)'Give I within limits.'	
ENDIF	
RETURN	
END	

Main program invoking FUNCTION	Main program calling SUBROUTINE
<pre> READ(*,*) A,B,C C C FUNCTION FX is invoked C ROOT1 = FX(1,A,B,C) ROOT2 = FX(2,A,B,C) WRITE(*,*) ROOT1,ROOT2 STOP END </pre>	<pre> READ(*,*) A,B,C C C SUBROUTINE is called C CALL ROOT(A,B,C,ROOT1,ROOT2) WRITE(*,*) ROOT1,ROOT2 STOP END </pre>

It is interesting to note that though only one **FUNCTION** subprogram is written to calculate both the roots, it can return only one value of it when invoked. On the other hand the **SUBROUTINE** does not suffer this limitation.

Layout of a program

It is a matter of preference to decide the order of placement of the main program and its subprograms. Actually the order does not matter, but, whatever the preference, it should be practiced consistently. Generally the main program is placed first followed by **SUBROUTINEs**, and the **FUNCTION** subprograms are placed after the subroutines.

EXTERNAL statement

The calling program is generally the main program, but a subprogram can also invoke or call another subprogram where the order of their placement does not matter. In such a case the main program cannot make a difference between a variable name and the name of the subprogram(s) that is/are being used by other subprogram(s) due to the reason that it actually has not directly used them. The way to tell the main program the difference is to place an **EXTERNAL** statement as the last declaration statement in the main program followed by the name(s) of the subprogram(s) that has/have been called by another subprograms. Its syntax is:

EXTERNAL FX,ROOT

where, **FX** and **ROOT** may be names of a **FUNCTION** and a **SUBROUTINE** subprogram, respectively, that have been called by another function or subroutine. The calling subprogram should have the name(s) of the called/invoked subprogram(s) in its argument list. For example, if the subroutine **ZEROES** uses the **FUNCTION** **FX** and the **SUBROUTINE** **ROOT**, it must have these two names in its argument list among others:

SUBROUTINE ZEROES(FX,ROOT, ...)

The same rule applies in case of a calling **FUNCTION** subprogram.

COMMON statement

It is possible to eliminate the dummy argument list from the subroutine, which is not possible in the case of a **FUNCTION** subprogram. A variable name used in a subroutine is not related to the same variable name in the calling program unless it is in the argument list of the subroutine. When it is desired to have a variable name in both the subroutine and the calling program sharing the same memory location, the **COMMON** statement comes in handy. The data required for processing and the results to be stored may be placed in **COMMON** storage available to both the calling program and the subprogram. In the case of a subroutine definition and subroutine **CALL** statement, the list of variables as its arguments can be placed separately in **COMMON** storage between the two programs.

A specific amount of memory is reserved for the running of a program. The compiler uses this block of memory by allocating it sequentially to the variables and arrays of the main program first and then to those of various subprograms. Usually there remains some unused memory after this allocation is completed. The **COMMON** storage area is this special segment of memory that starts at the end of the block and works forward, and it can be utilized by the **COMMON** statements in both the subprogram and its calling program.

This area is accessible to both the subprogram and its calling program or subprogram by the help of the **COMMON** statement together with the declaration of the names of variables that will share the sequential memory locations. Its syntax is as follows:

COMMON X,Y,Z,R1,R2,J

This should appear both in the calling program and the subprogram, though in the subprogram, names of the variables may appear differently, e.g.,

COMMON A,B,C,ROOT1,ROOT2,ICOUNT

In any case it is a must to maintain the order and type of the variables places in both the programs. Thus X and A, Y and B, Z and C, R1 and ROOT1, R2 and ROOT2 and J and ICOUNT will share the same memory locations.

The size of the array can be declared inside the **COMMON** statement, thus eliminating the **DIMENSION** statement. The following statement

DIMENSION X(100),Y(100),A(5,5)
COMMON X,Y,A

is equivalent to

COMMON X(100),Y(100),A(5,5)

In the construction of the list of variable names for the **COMMON** statement in a program or in various subprograms sharing the common storage locations, a consistency of mode should be maintained. It is explained by the following example:

Calling program

Subprogram

DIMENSION X(3),MM(30)
COMMON X,MM

DIMENSION I(25),J(10)
COMMON A,B,C,I,J

Thus X(1) is defined as A, X(2) as B, X(3) as C, the first 25 values of MM as I and the rest 10 values of MM as J, that is, the first 3 locations in common storage are treated as real numbers and the other 30 locations as integers.

When large arrays are used inside a **COMMON** statement, and a large number of subprograms are involved, it is advisable to set up more than one area of common storage. This is done by attributing an identifying name to the **COMMON** statement as follows:

COMMON /BLOK1/ X,Y,Z,R1,R2,I

This is called **labeled COMMON**. It creates a common storage area named **BLOK1** where the variables X, Y, Z, R1, R2 and I are stored in sequential manner. The name of the common storage area is user supplied, which goes by the same rules made for common FORTRAN variable names.

A single **COMMON** statement can be used to identify memory locations in both labeled and unlabeled **COMMONs**:

Calling program

Subprogram

COMMON X,Y,BLOK1/A(5),B(5)/BLOK2/I,J(10)

COMMON C,D/BLOK1/E(5),F(5)/BLOK2/MM,NN(10)

The variables X and Y in the calling program and C and D in the subprogram are declared under unlabeled **COMMON** whereas the rest are declared under labeled **COMMON** with identifying names **BLOK1** and **BLOK2**.

Arithmetic statement or line function

There is one simplification of **FUNCTION** subprograms that has a great deal in common with library functions. It is a single statement function that also returns one value to the point in the program where it is invoked. This version is known as **arithmetic statement function** or **line function** whereas the earlier one is called **multi-line function**. The statement function is defined in the program itself, before any declaration statements whatsoever and is used in the same manner as a library function. The name of a statement function is given the same way as to any FORTRAN variable. Thus the function name cannot exceed six characters, it should have a letter as the first character, its type (integer or real) is decided by the first letter type convention, etc. The integer type **FUNCTION** subprograms and arithmetic statement functions will yield integer type results and so on. It is declared using the following form:

$$\text{Name}(p_1, p_2, \dots, p_n) = \text{arithmetic expression}$$

where, $p_1, p_2, \dots, p_n$ are the parameters that should be single unsubscripted variables. The expression on the right hand side of the statement should not contain any subscripted variables. A statement function is internal to a program or a subprogram where it is defined and hence it can not be called by any other program or subprogram.

Since a statement function is internal to the program where it resides, it has an advantage that the **FUNCTION** subprogram does not offer. The variables that do not appear in the parameter list can be added in the statement itself. Thus

$$\text{FX}(X) = A * X ** 2 + B * X + C$$

is a correct arithmetic statement function. Here A, B, and C are not dummy variables because they are not included in the parameter list. Before the function is used, however, the values of these variables should be provided in the program.

An example program will be appropriate at this point to calculate the real root of any equation, be it quadratic or not by iterative method. Since FORTRAN will later be used to solve problems by numerical methods, this

early exposure to some of the important ideas, such as convergence, iteration, error limit, guessing, etc. will be helpful.

For comparison sake let us consider a quadratic equation, the real root of which will be estimated by an iterative method. The equation is given by

$$y = ax^2 + bx + c = 0 \quad 2.5$$

Now an expression for x can be obtained from this equation as

$$x = -\frac{a}{b} \cdot x^2 - \frac{c}{b} \quad 2.6$$

The right hand side of Eq. (2.6) should be known in order to calculate the value of x on its left hand side. Thus it can be rewritten as

$$f(x) = x_{i+1} = -\frac{a}{b} \cdot x_i^2 - \frac{c}{b} \quad 2.7$$

Initially, for $i=1$, the value of x_1 has to be supplied in order to calculate x_2 on the left hand side of the equation. This first value is unknown and hence a guess is to be made initially of the real root of Eq. (2.5). If this function $f(x)$ is put inside a **DO**-loop, and following the initial step, value of x_1 , that is, the first guess is replaced by x_2 , then in the next step x_2 by x_3 and so on, the successive values of x will approach the real root and the difference between any two successive values of x will reduce. In other words, this iteration scheme will converge to a real root of Eq. (2.5). It will really happen if the first guess is reasonable. A rough plot between x and $f(x)$ may provide a reasonable first approximation of the root.

In the program, the values of the coefficients a , b , and c along with the first guess can be supplied through a free format **READ** statement. The convergence criterion can be decided by testing (with a block **IF** statement) the value of the absolute difference between two consecutive values of x against a predetermined small value attributed to a variable called **ERROR**. It is obvious that the smaller the value of **ERROR** given the higher is the accuracy but at the cost of more iterations and hence more computer time. Usually a value in the order of 10^{-6} is moderate for **ERROR**.

The upper limit of the **DO**-loop should be sufficient for convergence. When the above-mentioned criterion for convergence is reached, the block **IF** with an **EXIT** statement should let the control go out of the loop. On the other hand a **DO WHILE** loop, if used, can eliminate the block **IF** as well as **EXIT** statement. We will examine both.

The function in Eq. (2.7) can be represented by a statement function **FX**.

Example program 2.13

```

C
C
C PROGRAM TO CALCULATE THE REAL ROOT OF A QUADRATIC EQUATION
C BY AN ITERATIVE METHOD.
C
C
C CHARACTER *1 S
C DATA ERROR /1.0E-06/
C
C THE ARITHMETIC STATEMENT FUNCTIONS FX AND FX1 ARE DEFINED
C
C      FX(X) = -A*X**2/B - C/B
C      FX1(X) = A*X**2 + B*X + C
C
10  WRITE(*,40)
    READ(*,50) S
    IF((S .EQ. 'N') .OR. (S .EQ. 'n')) STOP
    WRITE(*,60)
    READ(*,*) A,B,C
    WRITE(*,70)
20  READ(*,*) X0
C
    X1 = X0
    DO 30 I=1,1000
        N = I
        X2 = FX(X1)
        IF(ABS(X2 - X1) .LE. ERROR) EXIT
C
C THE NEW VALUE OF X IS GIVEN AS ITS OLD VALUE
C
        X1 = X2
30  CONTINUE
C
C THE VALUE OF FUNCTION IS CALCULATED WITH CALCULATED ROOT
C IF THIS VALUE IS LARGE, GO TO READ ANOTHER NEW GUESS
C
    VALUFX = FX1(X2)
    IF((ABS(VALUFX) .GT. ERROR) .OR. (I .EQ. 1001)) THEN
        WRITE(*,80) I,X0
        GOTO 20
    ENDIF
C
C ***** PRINT THE RESULTS *****

```

```

C
      WRITE(*,90) X2,N
C
C   GO TO READ NEW VALUES OF A, B, C
C
      GOTO 10
C
C   ***** FORMAT STATEMENTS *****
C
40  FORMAT(/12X,'Press [Y] to CONTINUE; [N] to STOP'//)
50  FORMAT(A1)
60  FORMAT(2X,'Enter the values of A, B and C'/)
70  FORMAT(2X,'Give the first guess'/)
80  FORMAT(2X,'Scheme is not converging. No. of iterations ',
1    ' ',I3,2X,'Give another guess other than ',F6.2//)
90  FORMAT(2X,'The root is calculated as ',E15.7,' after ',
1    ' ',I3,1X,'iterations.'//)
      END
C

```

The statement $N=I$ just after the **DO**-loop in the program is very important. From our knowledge of the **DO**-loop, we are aware that after a normal exit, the value of **I** outside the loop will be 1001, but the value of **N** will remain at 1000 because the control will not see the statement before going out of the loop. This technique is used especially when the counter variable is used in calculations outside the loop.

BLOCK DATA subprogram

The **DATA** statement is used in the program to store the value of **ERROR**. In the case of a large program involving a number of subprograms, the labeled **COMMON** statements may be used in the main program and in the subprograms. If some of the constants declared in the labeled **COMMON** are to be given values by **DATA** statement(s), some compilers may not permit it in labeled **COMMON** areas and some do not allow initialization on any **COMMON** areas. The **BLOCK DATA** subprogram defines initialization at the global level of the entire program. There should be only one such subprogram in a program, and often it is placed just after the main program and before any other subprogram. Its structure is as follows:

BLOCK DATA

Statements such as **DIMENSION**, labeled **COMMON** etc.

DATA statement(s)

END

The **BLOCK DATA**, like any other subprogram, can be given a name, which it is optional. Since its name is not referred to by any program or subprogram, we would prefer to omit it.

ENTRY statement

This statement helps use a part of a subprogram. Thus it defines a subprogram within a subprogram. When a subprogram is normally evoked, the control goes to the first executable statement followed by the subprogram defining statement. The instruction to enter a subprogram at different entry points is given by the following nonexecutable statement:

ENTRY n(parameter list)

where, **n** is the name assigned to the entry point, which should be different from the name of the subprogram. The parameter list is optional. The **ENTRY** statement does not have any effect on the logic of the subprogram. The name assigned to the entry point is called by the other program or subprogram just as calling a **SUBROUTINE**. When called, the program execution begins at the entry point and proceeds until a **RETURN** or an **END** statement is encountered.

Variable Input/Output file name

Sometime it is desirable to read data from an existing file or to write to one. When input data are entered through the terminal, the same should not be lost after the program is run, rather these should be stored in an input file created by the user at that moment. This can be done in FORTRAN using the same **OPEN** statement learned earlier to open an external input or output file. The only difference here is to use a character variable as the name of **FILE** as one of the parameters in the argument list of **OPEN** statement, which should not be declared under apostrophes. The **CHARACTER** statement should define the length of the file name denoted by the character variable. The following example program (2.14) explains this special feature.

REWIND statement

In this context it will be good to explain the file-positioning statement **REWIND**. This instruction positions the index for reading the content of a

file at the beginning of it. When data are to be read more than once in the same program this statement is used. Its use will be observed in the example program 2.15 and in regression programs given in the next part. If a file is already rewound, invoking this statement does not affect or alter anything. The syntax of this statement is given below:

```
50 READ(10,*) N,(X(I),I=1,N)
    statement(s)
    REWIND 10
    GOTO 50
```

Thus, **REWIND** can work only with external input files.

PAUSE statement

When a compiled program is executed it may be necessary to examine the results before proceeding with its normal execution. The simultaneous use of the control key and s-letter key to achieve this is not very convenient. The **PAUSE** statement causes a temporary halt of the program execution and pressing the Return or Enter key allows the execution to continue. It can be placed inside a program where required and after compilation becomes a part of the program. Its format is as follows:

PAUSE statement between single apostrophes

where the statement will appear on the screen and the message "Press <Enter> to continue" will appear on the line below it.

Metacommand \$INCLUDE

This statement directs the compiler to proceed as though the content of source file name specified in it were inserted in the program at the point where **\$INCLUDE** is placed. At the end of the included file, the compiler resumes processing the original source file at the line following this Metacommand. Its format is as follows:

\$INCLUDE:'filename'

This Metacommand should be placed on the first column of the program with the source code. The filename is a valid file specification as described for the disk operating system (DOS). It can precede with the letter for the destination drive, such as **A**, **B**, **C**, **D**, etc. and the subdirectory name. Thus

\$INCLUDE:'C:\FORTRAN\PSYCRO'

will include the content of the file named **PSYCRO**, which is in the **FORTRAN** subdirectory in drive **C**. This command helps in joining several programs or program segments without physically bringing them together and thus curbing the length of a single program. The regression programs given in the next part shows its application.

Example program 2.14

```

C _____
C
C PROGRAM TO CREATE AN INPUT DATA FILE
C
C      DIMENSION X(10,100),Y(100)
C      CHARACTER *18 FI
C      CHARACTER *1 S
C
C      FI : VARIABLE NAME FOR INPUT FILE
C
C      WRITE(*,30)
C      READ(*,40) S
C      IF((S.EQ. 'Y') .OR. (S.EQ. 'y')) THEN
C          WRITE(*,50)
C          READ(*,60) FI
C          OPEN(10,FILE=FI,STATUS='NEW')
C          GOTO 10
C      ELSE
C          WRITE(*,70)
C          STOP
C      ENDIF
C
C      ***** TYPE IN INPUT DATA *****
C
10  WRITE(*,80)
    READ(*,*) N,NVAR,(Y(I),I=1,N),((X(I,J),J=1,N),I=1,NVAR)
    WRITE(10,90) N,(Y(I),I=1,N),((X(I,J),J=1,N),I=1,NVAR)
    WRITE(*,100)
    READ(*,40) S
    IF((S.EQ. 'Y') .OR. (S.EQ. 'y')) GOTO 10
20  WRITE(*,110) FI
    CLOSE(10,STATUS='KEEP')
C

```

```

C      ***** FORMAT STATEMENTS *****
C
30  FORMAT(/2X,'Type [Y] to READ data from SCREEN and <ENTER>.'/
    1      2X,'Type [N] and press <ENTER> to QUIT.'//)
40  FORMAT(A1)
50  FORMAT(/2X,'Specify INPUT file within 18 characters to TYPE',
    1      ' DATA from SCREEN and <ENTER>.'/)
60  FORMAT(A18)
70  FORMAT(/2X,'Thank you for using this program. BYE!')
80  FORMAT(/1X,60('-')/2X,'Give N,NVAR,Y-values,X-values and ',
    1      '<ENTER>.'//2X,'N: No. of DATA POINTS; NVAR: No. of ',
    2      'INDEPENDENT variables'/1X,60('-')/)
90  FORMAT(1X,I3/5(1X,E15.8))
100 FORMAT(/2X,'If have more DATA, press [Y] and <ENTER>.'/2X,
    1      'When all DATA exhausted, press [N] and <ENTER>.'/)
110 FORMAT(2X,'You have created an INPUT file ',A18/2X,'All ',
    1      'your DATA (N, Y-values, X-values) are stored in it.'/)
      STOP
      END

```

C

In the following example program, the use of subprograms will be demonstrated and at the same time, the programs that have been developed so far will be joined together showing the mechanism of turning them into different subprograms.

Here the main program will calculate the roots of a quadratic equation by the common analytical method (example program 2.11) and by the method of iteration (example program 2.13). It will also calculate the standard deviation and arithmetic average of a given number of data points (example program 2.12). One **SUBROUTINE** will calculate the real roots by analytical method and the other **SUBROUTINE** will follow the iterative method to calculate the real root. Statistical calculations in the second part will be carried out by a **FUNCTION** subprogram. Instead of using two arithmetic statement functions — one each for analytical expression of the real roots, a **FUNCTION** subprogram will be used. The first subroutine will use this function and hence an **EXTERNAL** statement will be required in the main program. In the second subroutine, an arithmetic statement function will represent the functional expression of the iterative scheme and another the original quadratic function.

Example program 2.15

C

C

C THIS IS A MAIN PROGRAM TO COMPUTE REAL ROOTS OF A QUADRATIC

C EQUATION BY ANALYTICAL AND ITERATIVE METHODS AND COMPARE THE
C RESULTS. IT ALSO CALCULATES THE ARITHMETIC MEAN AND STANDARD
C DEVIATION OF A NUMBER OF DATA POINTS.

C
C

```
COMMON /BLOK1/A,B,C,X0,R1,R2,R,COUNT
DIMENSION X(100)
INTEGER COUNT
REAL MEAN
CHARACTER *1 S
EXTERNAL FX
```

C

```
OPEN(10,FILE='DATA.IN',STATUS='OLD')
OPEN(20,FILE='DATA.OUT',STATUS='NEW')
```

C

```
DO 50 MM=1,100
  WRITE(*,70)
  READ(*,80) S
  IF((S .EQ. 'N') .OR. (S .EQ. 'n')) EXIT
  WRITE(*,90)
  READ(*,*) KK
  IF((KK .EQ. 1) .OR. (KK .EQ. 2)) THEN
```

C

C START A DO-LOOP TO READ DATA SEVERAL TIMES

C

```
  WRITE(*,100)
  READ(*,*) N
  DO 20 I=1,N
```

C

```
    WRITE(*,110)
    READ(*,*) A,B,C
    D = B**2 - 4.0*A*C
    IF(D .LT. 0.0) THEN
      WRITE(*,120)
      READ(*,*) A,B,C
    ELSEIF(A .EQ. 0.0) THEN
      WRITE(*,130)
      READ(*,*) A,B,C
    ENDIF
```

C

```
    IF(KK .EQ. 1) THEN
```

C

C SUBROUTINE IS CALLED TO CALCULATE ROOTS BY ANALYTICAL METHOD

C

```
      CALL ROOT(FX)
```

```

WRITE(*,140) R1,R2
WRITE(20,140) R1,R2
ELSEIF(KK .EQ. 2) THEN
C
C SUBROUTINE IS CALLED TO CALCULATE ROOT BY ITERATIVE METHOD
C
WRITE(*,150)
10 READ(*,*) X0
CALL ROOT1
IF(R .EQ. 0.0) THEN
WRITE(*,160) COUNT,X0
GOTO 10
ELSE
WRITE(*,170) R,COUNT
WRITE(20,170) R,COUNT
ENDIF
ENDIF
C
C GO BACK TO INITIAL DO STATEMENT
C
20 CONTINUE
ELSEIF(KK .EQ. 3) THEN
C
C DATA ARE READ; NUMBER OF DATA POINTS CAN NOT EXCEED 100
C
DO 30 J=1,100
READ(10,*,END=40) COUNT,(X(I),I=1,COUNT)
MEAN = STAT(1,X)
STDEV = STAT(2,X)
WRITE(*,180) COUNT,STDEV,MEAN
C
C WRITE THE RESULT ALSO IN THE OUTPUT FILE 'DATA.OUT'
C
WRITE(20,180) COUNT,STDEV,MEAN
C
C IF THERE ARE MORE DATA GO BACK TO READ THEM, OTHERWISE STOP
C
30 CONTINUE
ENDIF
C
40 REWIND 10
50 CONTINUE
C
C CLOSE BOTH EXTERNAL FILES
C

```

```

60  CLOSE(10,STATUS='KEEP')
    CLOSE(20,STATUS='KEEP')
C
C  *****  FORMAT STATEMENTS  *****
C
70  FORMAT(/2X,'If want to CONTINUE, Press [Y], else [N] ',
1      'and <ENTER>.'/)
80  FORMAT(A1)
90  FORMAT(2X,'This program has multiple capabilities.//
1      2X,'Type [1] to compute ROOTS by analytical method'
2      /2X,'Type [2] to compute ROOT by iterative method'/
3      2X,'Type [3] to compute standard deviation and ',
4      'mean of data points'//)
100 FORMAT(/10X,'Give the number of quadratic equations'//)
110 FORMAT(2X,'Give values of A, B and C'/)
120 FORMAT(2X,'Root of a negative number. Give A, B & C'/)
130 FORMAT(2X,'A can not be ZERO. Give other A, B & C'/)
140 FORMAT(2X,'The value of the FIRST root = ',1PE15.7/2X,
1      'The value of the SECOND root = ',1PE15.7//)
150 FORMAT(2X,'Give the first GUESS'/)
160 FORMAT(2X,'Scheme is not converging. No. of iterations ',
1      '= ',I3/2X,'Give another guess other than ',F6.2//)
170 FORMAT(2X,'The root is calculated as ',E15.7,' after ',
1      I3,I3X,'iterations.//)
180 FORMAT(2X,'Standard Deviation among ',I3,' Data points ',
1      'is : ',F10.6/2X,'Mean = ',F10.6//)
    STOP
    END
C
C
C  SUBROUTINE TO CALCULATE ROOTS BY ANALYTICAL METHOD
C
C
C      SUBROUTINE ROOT(FX)
C      COMMON /BLOK1/X,Y,Z,X0,R1,R2,R,COUNT
C      INTEGER COUNT
C
C      R1 = FX(1,X,Y,Z)
C      R2 = FX(2,X,Y,Z)
C
C      RETURN
C      END
C
C
C  SUBROUTINE TO CALCULATE THE REAL ROOT OF A QUADRATIC EQUATION

```

```

C   BY AN ITERATIVE METHOD.
C
C
      SUBROUTINE ROOT1
      COMMON /BLOK1/F,G,H,X0,R1,R2,X2,COUNT
      INTEGER COUNT
      DATA ERROR /1.0E-06/

C
C   THE ARITHMETIC STATEMENT FUNCTIONS FX0 AND FX1 ARE DEFINED
C
      FX0(X) = -F*X**2/G - H/G
      FX1(X) = F*X**2 + G*X + H

C
      X1 = X0
      DO 10 I=1,1000
          COUNT = I
          X2 = FX0(X1)
          IF(ABS(X2 - X1) .LE. ERROR) EXIT

C
C   THE NEW VALUE OF X IS GIVEN AS ITS OLD VALUE
C
      X1 = X2
10   CONTINUE

C
C   THE VALUE OF FUNCTION IS CALCULATED WITH CALCULATED ROOT
C   IF THIS VALUE IS LARGE GO TO READ ANOTHER NEW GUESS
C
      VALUFX = FX1(X2)
      IF((ABS(VALUFX) .GT. ERROR) .OR. (I.EQ. 1001)) THEN
          X2 = 0.0
          RETURN
      ENDIF

C
      RETURN
      END

C
C
C   FUNCTION TO CALCULATE STANDARD DEVIATION OF ONE OR MORE
C   SET OF DATA POINTS.
C
C
      FUNCTION STAT(L,X)
      DIMENSION X(100)
      COMMON /BLOK1/A,B,C,X0,R1,R2,R,COUNT
      REAL MEAN

```

```

      INTEGER COUNT
C
      MEAN = 0.0
      STDEV = 0.0
      DO 10 I=1,COUNT
          MEAN = MEAN + X(I)/FLOAT(COUNT)
10  CONTINUE
C
C  NOW THE LOOP FOR CALCULATION OF STANDARD DEVIATION
C
      DO 20 I=1,COUNT
          STDEV = STDEV + (X(I) - MEAN)**2/FLOAT(COUNT - 1)
20  CONTINUE C
      STDEV = SQRT(STDEV)
      IF(L .EQ. 1) THEN
          STAT = MEAN
      ELSEIF(L .EQ. 2) THEN
          STAT = STDEV
      ENDIF
C
      RETURN
      END
C


---


C
C  FUNCTION TO CALCULATE TWO REAL ROOTS OF A QUADRATIC EQUATION
C


---


C
      FUNCTION FX(I,A,B,C)
C
      D = B**2 - 4.0*A*C
      IF(I .EQ. 1) THEN
          FX = (-B + SQRT(D))/2.0/A
      ELSE
          FX = (-B - SQRT(D))/2.0/A
      ENDIF
      RETURN
      END
C


---



```

In the above example program the use of declaration statements, such as block **COMMON**, **DIMENSION**, **REAL**, **INTEGER**, **CHARACTER**, **DATA**, and **EXTERNAL**; library functions **FLOAT**, **SQRT**, and **ABS**; block **IF**, **DO**-loop and **EXIT** statements; **OPEN**, **CLOSE** and **REWIND** a file; subprograms, such as, **FUNCTION** and **SUBROUTINE**; and arithmetic statement functions, etc. has been demonstrated. In the whole program there

is only one **GOTO** statement, which could not be otherwise avoided. The reader should study this program in detail to learn applications of various features learned so far.

In all the example programs it is very important to note the following features:

- (i) All **DO**-loops and block **IF**s are indented for better clarity,
- (ii) Statement numbers start with 10 and proceed with a step of 10,
- (iii) All **FORMAT** statements are grouped in one place at the end of the program,
- (iv) Floating point numbers without any decimal part have been written with a zero after the decimal point,
- (v) Use of **GOTO** statements has been kept at minimum,
- (vi) Enough comments have been used to understand the program objective(s),
- (vii) Mixing of real data with integers and vice versa has been avoided using statements like **REAL**, **INTEGER**, **FLOAT** and **IFIX**, and
- (viii) The main program has been positioned first followed by **SUBROUTINE**s and **FUNCTION**s thereafter.

The FORTRAN language does not compel its users to follow the features stated above, but it is important that the users should not forget their responsibility to use this powerful and friendly language in the most disciplined and structured way that they can.

Table 2.2 Keywords in FORTRAN.

ABS	ACCESS	ACOS	ALOG	ALOG10
AMAX0	AMAX1	AMIN0	AMIN1	AMOD
AND	ASIN	ASSIGN	ATAN	ATAN2
BACKSPACE	BLANK	BLOCK	CALL	CHAR
CHARACTER	CLOSE	COMMON	COMPLEX	CONTINUE
COS	COSH	DABS	DACOS	DASIN
DATA	DATAN	DATAN2	DCOS	DCOSH
DEXP	DIMENSION	DLOG	DLOG10	DMAX1
DMIN1	DMOD	DO	DOUBLE	DSIN
DSINH	DSQRT	DTAN	DTANH	ELSE
ELSEIF	END	ENDDO	ENDFILE	ENDIF
ENDWHILE	ENTRY	EQUIVALENCE	EQV	ERR
EXIST	EXIT	EXP	EXPLICIT	EXTERNAL
FALSE	FILE	FLOAT	FMT	FOR
FORM	FORMAT	FUNCTION	GOTO	IABS
ICHAR	IF	IFIX	IMPLICIT	INCLUDE
INDEX	INPUT	INQUIRE	INTEGER	INTRINSIC
IOSTAT	KEEP	LEN	LOGICAL	MAX
MAX0	MIN	MIN0	MOD	NAMED
NEW	NOT	NULL	NUMBER	OLD
OPEN	OPENED	OR	PARAMETER	PAUSE
PRINT	PROGRAM	PUNCH	READ	REAL
REC	RECL	RETURN	REWIND	SAVE
SCRATCH	STATUS	SIN	SINH	SQRT
STOP	SUBROUTINE	TAN	TANH	TRUE
TYPE	UNKNOWN	WHILE	WRITE	ZERO

References

1. **Davis, G. B. and Hoffmann, T. R.**, *FORTRAN A Structured, Disciplined Style*. McGraw-Hill Book Co., New York, 1978.
2. **International Business Machine Corporation**, FORTRAN Compiler. IBM Computer Language Series, version 2.00, 1984.
3. **Lipschutz, S. and Poe, A.**, *Programming with FORTRAN Including Structured FORTRAN*. Schaum's Outline Series in Computers, McGraw-Hill Book Co., New York, 1978.
4. **Rule, W. P.**, *FORTRAN : A Practical Approach with Style and Structure*. Prindle, Weber & Schmidt, Boston, 1980.

PART III

NUMERICAL METHODS

CHAPTER 3.1

Numerical Differentiation

Differentiation is a process in calculus used to express the rate of change of a given function with respect to location or time. If there is a function $f(x)$ that is analytic in the given range of the independent variable x , the first derivative of the function, denoted by $f'(x)$ is defined by

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad 3.1.1$$

When the infinitesimal increment in x , denoted by h , tends to zero (actually it should not be exactly zero, rather by definition should be a very small quantity), the change in the value of the function between the interval h , divided by this increment is called the **first derivative** of the original function.

The rules of differentiation are established and very well defined in calculus. The only problem is that there are so many of these rules and that they are so dependent on the particular function that an engineer who does not have the background of a mathematician gets lost when an unknown function appears whose derivative is to be evaluated. The numerical differentiation employs only one general method with simple arithmetic operations, which can easily be implemented in a digital computer, thus making the engineer's work easy.

Taylor Series expansion

The general method that is employed to derive different expressions for calculating derivatives (first, second, third, etc.) calls for understanding the foundation of numerical methods known as the **Taylor Series**. It is an infinite power series to approximate the value of a function in a region where the function is analytic. A function is said to be analytic in a region of the variable where the function $f(x)$ and its derivatives must exist and yield finite values. Thus a function $f(x) = 1/x$ is not analytic for an extremely small value of x for which $f(x)$ becomes so large that it represents infinity for any practical purpose. In other words, if the value of the function $f(x)$ can be expressed by the following infinite power series in the region $x = a$,

$$f(x) = f(a) + (x - a).f'(a) + \frac{(x - a)^2}{2!}.f''(a) + \frac{(x - a)^3}{3!}.f'''(a) + \dots \quad 3.1.2$$

then $f(x)$ is said to be analytic in the same region of the variable x , and the infinite series is called the Taylor Series expansion of $f(x)$ in the neighborhood of $x = a$.

If the series expressed by Eq. (3.1.2) is convergent, then the first few terms of the series approximate the value of the function quite reasonably, provided the difference $(x - a)$ remains sufficiently small. Since in most of the derivations of numerical formulas only a few terms of the Taylor Series is taken into account, it is very important that $(x - a)$ remains a small positive fraction, i.e., less than one. In general this difference can be larger than one, but within the scope of our studies we neglect it.

It is very important to understand the Taylor Series at this point and an example will help. Let us expand e^x about $x = 0$ and then approximate the value of e^x for a very small value and then a larger value of x .

$$\begin{aligned} e^x &= f(x - 0) \\ &= e^0 + (x - 0).e^0 + \frac{(x - 0)^2}{2!}.e^0 + \frac{(x - 0)^3}{3!}.e^0 + \dots \\ &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots \end{aligned}$$

Now, for a fractional value of x , say 0.01, the exact value is $e^{0.01} = 1.0100501667$. If we consider the first four terms of the series,

$$\begin{aligned} e^{0.01} &= 1 + 0.01 + 0.00005 + 0.0000001667 \\ &= 1.0100501667 \end{aligned}$$

this is accurate up to the tenth decimal place. On the other hand, if x is taken equal to 1.01, the exact value is $e^{1.01} = 2.745601015$. Again considering first four terms of the series,

$$\begin{aligned} e^{1.01} &= 1 + 1.01 + 0.51005 + 0.171716833 \\ &= 1.691766833 \end{aligned}$$

which is far from the exact value. This example demonstrates the fact just discussed that it is important to keep the difference $(x - a)$ to a very small positive fraction when the function is approximated by only the first few terms of the series.

Finite difference formulas for differentiation

Now let us use the Taylor Series expansion taking different numbers of terms to derive expressions for differentiation. The left hand side term of the Taylor Series expansion in Eq. (3.1.2) can be written as $f(x) = f[a + (x - a)]$. Now if the difference $(x - a)$, which according to our understanding should be a small positive fraction, is denoted by h , $f(x)$ can be expressed as $f(a + h)$ and the Taylor Series expansion as

$$f(a + h) = f(a) + h \cdot f'(a) + \frac{h^2}{2!} \cdot f''(a) + \frac{h^3}{3!} \cdot f'''(a) + \frac{h^4}{4!} \cdot f^{iv}(a) + \dots$$

In the same token, $f(x + h)$ can be expressed as

$$f(x + h) = f(x) + h \cdot f'(x) + \frac{h^2}{2!} \cdot f''(x) + \frac{h^3}{3!} \cdot f'''(x) + \frac{h^4}{4!} \cdot f^{iv}(x) + \dots \quad 3.1.3$$

Rearranging Eq. (3.1.3),

$$f'(x) = \frac{f(x + h) - f(x)}{h} - \frac{h}{2} \cdot f''(x) - \frac{h^2}{6} \cdot f'''(x) - \frac{h^3}{24} \cdot f^{iv}(x) - \dots$$

If the terms having h and its higher powers are neglected, we get

$$f'(x) = \frac{f(x + h) - f(x)}{h} \quad 3.1.4$$

When Eq. (3.1.4) is compared with Eq. (3.1.1), which defines the first derivative, they look strikingly similar. Thus the numerical expression given by Eq. (3.1.4) turns out to be Eq. (3.1.1) when h approaches zero, that is, for all practical purposes, when it is taken sufficiently small.

The expression for the first derivative of $f(x)$ with respect to x given by Eq. (3.1.4) is accurate to within an error order of h and is known as the **first-order forward difference formula**. It was obtained by forward Taylor Series expansion.

Similarly, performing a backward Taylor Series expansion of $f(x - h)$, we get

$$f(x - h) = f(x) - h \cdot f'(x) + \frac{h^2}{2!} \cdot f''(x) - \frac{h^3}{3!} \cdot f'''(x) + \frac{h^4}{4!} \cdot f^{iv}(x) + \dots \quad 3.1.5$$

and the expression of $f'(x)$ is obtained as

$$f'(x) = \frac{f(x) - f(x - h)}{h} + \frac{h}{2} \cdot f''(x) - \frac{h^2}{6} \cdot f'''(x) + \frac{h^3}{24} \cdot f^{iv}(x) - \dots$$

Neglecting the terms with h and its higher powers,

$$f'(x) = \frac{f(x) - f(x - h)}{h} \quad 3.1.6$$

This is a backward difference expression for the first derivative of $f(x)$ with respect to x . It is also a first-order formula since all the terms starting with the first power of h are neglected.

If Eq. (3.1.5) is subtracted from Eq. (3.1.3), we get, after rearranging,

$$f'(x) = \frac{f(x + h) - f(x - h)}{2h} - \frac{h^2}{3} \cdot f'''(x) - \frac{h^4}{60} \cdot f^{iv}(x) - \dots$$

Neglecting the terms with h^2 and higher powers of h ,

$$f'(x) = \frac{f(x + h) - f(x - h)}{2h} \quad 3.1.7$$

Eq. (3.1.7) is the central difference expression for the first derivative of $f(x)$ with respect to x . It is a second-order formula because the terms starting from the second power of h are not taken into account in it.

Usually when the functional form is available for differentiation or when the data points are sufficient to evaluate $f(x+h)$ and $f(x-h)$, the central difference form is preferred because of its one-order higher accuracy than the forward and backward difference formulas given by Eqs. (3.1.4) and (3.1.6), respectively.

There are still higher-order finite difference formulas for $f'(x)$ as given in Tables 3.1.3 and 3.1.4; nevertheless, the mostly used forms are the ones developed here and given in Tables 3.1.1 and 3.1.2. In fact, the Richardson

extrapolation formula is utilized in higher-than-first-order formulas to partially remove the truncation error.

Table 3.1.1. First-order Forward and Backward Difference Representations.

	f_{n-4}	f_{n-3}	f_{n-2}	f_{n-1}	f_n	f_{n+1}	f_{n+2}	f_{n+3}	f_{n+4}
<u>Forward</u>									
$h.f'(x_n) =$					-1	1			
$h^2.f''(x_n) =$					1	-2	1		
$h^3.f'''(x_n) =$					-1	3	-3	1	
$h^4.f^{iv}(x_n) =$					1	-4	6	-4	1
<u>Backward</u>									
$h.f'(x_n) =$				-1	1				
$h^2.f''(x_n) =$			1	-2	1				
$h^3.f'''(x_n) =$		-1	3	-3	1				
$h^4.f^{iv}(x_n) =$	1	-4	6	-4	1				

Table 3.1.2. Second-order Central Difference Representations.

	f_{n-2}	f_{n-1}	f_n	f_{n+1}	f_{n+2}
$2h.f'(x_n) =$		-1	0	1	
$h^2.f''(x_n) =$		1	-2	1	
$2h^3.f'''(x_n) =$	-1	2	0	-2	1
$h^4.f^{iv}(x_n) =$	1	-4	6	-4	1

Table 3.1.3. Second-order Forward and Backward Difference Representations.

	f_{n-5}	f_{n-4}	f_{n-3}	f_{n-2}	f_{n-1}	f_n	f_{n+1}	f_{n+2}	f_{n+3}	f_{n+4}	f_{n+5}
<u>Forward</u>											
$2h.f'(x_n) =$						-3	4	-1			
$h^2.f''(x_n) =$						2	-5	4	-1		
$2h^3.f'''(x_n) =$						-5	18	-24	14	-3	
$h^4.f^{iv}(x_n) =$						3	-14	26	-24	11	-2
<u>Backward</u>											
$2h.f'(x_n) =$				1	-4	3					
$h^2.f''(x_n) =$			-1	4	-5	2					
$2h^3.f'''(x_n) =$		3	-14	24	-18	5					
$h^4.f^{iv}(x_n) =$	-2	11	-24	26	-14	3					

Table 3.1.4. Fourth-order Central Difference Representations.

	f_{n-3}	f_{n-2}	f_{n-1}	f_n	f_{n+1}	f_{n+2}	f_{n+3}
$12h.f'(x_n) =$		1	-8	0	8	-1	
$12h^2.f''(x_n) =$		-1	16	-30	16	-1	
$8h^3.f'''(x_n) =$	1	-8	13	0	-13	8	-1
$6h^4.f^{iv}(x_n) =$	-1	12	-39	56	-39	12	-1

Richardson extrapolation

If the error in certain formula approximating $f^n(x)$ is $C.h^r.f^r(\xi)$, where C is a constant, ξ lies between the two limits of x , and, r is a positive integer, two calculations can be performed — one with h_1 and the other with h_2 to calculate $f_1^n(x)$ and $f_2^n(x)$, respectively. Thus the true value of the operation (in this case it is differentiation), $f^n(x)$ can be expressed as

$$f^n(x) = f_1^n(x) + C.h_1^r.f^r(\xi_1) \quad 3.1.8$$

and

$$f^n(x) = f_2^n(x) + C.h_2^r.f^r(\xi_2) \quad 3.1.9$$

Now, letting $f^r(\xi_1) = f^r(\xi_2)$, from Eqns. (3.1.8) and (3.1.9),

$$\frac{f^n(x) - f_1^n(x)}{f^n(x) - f_2^n(x)} = \left(\frac{h_1}{h_2} \right)^r$$

Subtracting 1 from both the sides and rearranging,

$$f^n(x) = f_2^n(x) + \frac{f_2^n(x) - f_1^n(x)}{\left(\frac{h_1}{h_2} \right)^r - 1}$$

When $h_2 = \frac{h_1}{2}$, the above expression becomes

$$f^n(x) = f_2^n(x) + \frac{f_2^n(x) - f_1^n(x)}{2^r - 1} \quad 3.1.10$$

In the case of the central difference formula derived in this text, the value of r is 2.

This type of extrapolation is very effective in the case of the numerical integration as we will see in the next chapter. In fact, the Richardson extrapolation concept was developed for smoothing the truncation error in the integration formulas.

Let us now develop numerical expressions for higher derivatives of $f(x)$ with respect to x . To do this we will further need backward Taylor Series expansions of $f(x+2h)$, $f(x+3h)$, $f(x-2h)$ and $f(x-3h)$ besides the ones already developed (Eqs. 3.1.3 and 3.1.5).

Now rewriting Eqs. (3.1.3) and (3.1.5) and adding the expansions for $f(x+2h)$, $f(x+3h)$, $f(x-2h)$ and $f(x-3h)$,

$$f(x+h) = f(x) + h.f'(x) + \frac{h^2}{2!}.f''(x) + \frac{h^3}{3!}.f'''(x) + \frac{h^4}{4!}.f^{iv}(x) + \dots \quad 3.1.11$$

$$f(x+2h) = f(x) + 2h.f'(x) + \frac{4h^2}{2!}.f''(x) + \frac{8h^3}{3!}.f'''(x) + \frac{16h^4}{4!}.f^{iv}(x) + \dots \quad 3.1.12$$

$$f(x+3h) = f(x) + 3h.f'(x) + \frac{9h^2}{2!}.f''(x) + \frac{27h^3}{3!}.f'''(x) + \frac{81h^4}{4!}.f^{iv}(x) + \dots \quad 3.1.13$$

$$f(x-h) = f(x) - h.f'(x) + \frac{h^2}{2!}.f''(x) - \frac{h^3}{3!}.f'''(x) + \frac{h^4}{4!}.f^{iv}(x) + \dots \quad 3.1.14$$

$$f(x-2h) = f(x) - 2h.f'(x) + \frac{4h^2}{2!}.f''(x) - \frac{8h^3}{3!}.f'''(x) + \frac{16h^4}{4!}.f^{iv}(x) + \dots \quad 3.1.15$$

$$f(x-3h) = f(x) - 3h.f'(x) + \frac{9h^2}{2!}.f''(x) - \frac{27h^3}{3!}.f'''(x) + \frac{81h^4}{4!}.f^{iv}(x) + \dots \quad 3.1.16$$

Multiplying Eq. (3.1.11) by 2 and subtracting it from Eq. (3.1.12) and after rearranging,

$$f''(x) = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} - h.f'''(x) - \frac{7h^2}{12}.f^{iv}(x) - \dots$$

Neglecting the terms with h and its higher powers, the first-order forward difference formula for the second derivative is given by:

$$f''(x) = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} \quad 3.1.17$$

Again multiplying Eq. (3.1.14) by 2 and subtracting it from Eq. (3.1.15) and after rearranging,

$$f''(x) = \frac{f(x) - 2f(x-h) + f(x-2h)}{h^2} + h \cdot f'''(x) - \frac{7h^2}{12} \cdot f^{iv}(x) + \dots$$

Neglecting the terms with h and its higher powers, the first-order backward difference formula for the second derivative is obtained as:

$$f''(x) = \frac{f(x) - 2f(x-h) + f(x-2h)}{h^2} \quad 3.1.18$$

To obtain the central difference formula of the second derivative, Eq. (3.1.11) is added to Eq. (3.1.14) finally to obtain

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2}{12} \cdot f^{iv}(x) - \dots$$

Neglecting the terms with h^2 and its higher powers, the second-order central difference formula for the second derivative becomes

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \quad 3.1.19$$

The other derivations can be made in similar manner using the Taylor Series expansions, some of which are expressed by Eqs. (3.1.11) through (3.1.16). In order to facilitate the derivation of Open and Closed formulas for the solution of ordinary differential equations described in Chapter 3.6, only one more derivation of the backward difference formula for the third derivative of $f(x)$ with respect to x will be presented here.

To obtain the backward difference formula for the third derivative, first Eqs. (3.1.14) and (3.1.15) are multiplied by 3 and the former is subtracted from the latter, then Eq. (3.1.16) is subtracted from the resultant finally to give

$$f'''(x) = \frac{f(x) - 3f(x-h) + 3f(x-2h) - f(x-3h)}{h^3} + \frac{3h}{2} \cdot f^{iv}(x) - \dots$$

Neglecting the terms with h and its higher powers, the first-order backward difference formula for the third derivative becomes

$$f''''(x) = \frac{f(x) - 3f(x-h) + 3f(x-2h) - f(x-3h)}{h^3} \quad 3.1.20$$

For convenience let us denote $f(x)$, $f(x+h)$, $f(x+2h)$, $f(x+3h)$, $f(x+4h)$, $f(x+5h)$, $f(x-h)$, $f(x-2h)$, $f(x-3h)$, $f(x-4h)$, and $f(x-5h)$ as f_n , f_{n+1} , f_{n+2} , f_{n+3} , f_{n+4} , f_{n+5} , f_{n-1} , f_{n-2} , f_{n-3} , f_{n-4} , and f_{n-5} , respectively. The first-order forward and backward difference representations up to the fourth derivative are shown in Table 3.1.1. The second-order central difference representations up to the fourth derivative are shown in Table 3.1.2. The other higher-order representations are given in Tables 3.1.3 and 3.1.4 (Hornbeck, 1975).

In this first chapter on numerical methods, it is necessary to mention about analysis of the error. Without a knowledge about different types of errors occurred during computations, it is difficult to rely upon an answer produced by a computer. This subject has been touched in the last chapter, namely 3.8. The reason for presenting it at the end lies in the fact that after acquiring sufficient knowledge of numerical methods it will be easy to discuss errors relevant to particular methods already discussed.

Numerical differentiation consists of subtracting two very close numbers and dividing also by a very small number. It is thus very sensitive to round-off error. A double precision arithmetic is always recommended for numerical differentiation. It is relatively easier and more reliable to perform numerical integration than numerical differentiation. Analytically, on the other hand, differentiation is much easier than integration. We will see later in this chapter that numerical methods are sometimes not reliable in differentiating trigonometric functions. Thus caution should be taken before accepting results on differentiation by any of the methods discussed here.

A program has been developed in FORTRAN for numerical differentiation of any function that is analytic in the region of interest of the independent variable. It has two subroutines **FINITE** and **FINIT**, which employ normal- and higher-order methods, respectively, the finite difference representations of which are given in Tables 3.1.1 through 3.1.4. The function to be differentiated is supplied by the user through an external FORTRAN program named **FCTN.FOR**. The source code listings of the subroutines **FINITE(F,I)** and **FINIT(F,I)** are given in Appendix A.3.1.1 and A.3.1.2, respectively. A sample program listing of **FCTN.FOR** is given in Appendix A.3.1.3.

The main program is written in such a way that it offers its users the choice of both the normal- and higher-order methods to differentiate a function. The Richardson extrapolation formula has been used for the central difference scheme of normal-order and for all the schemes of the higher-order methods.

When the compiled program is executed, it asks first for an option on the first, second, third, or fourth derivative. Once opted, it again asks about the scheme, namely whether the user wants the forward, backward, or central

difference scheme. Lastly, it offers an option to choose the normal-order methods discussed in this chapter and given in Tables 3.1.1 and 3.1.2 or the higher-order representations given in Tables 3.1.3 and 3.1.4. The results show all the options, value of the required derivative, value at which it is evaluated, and the step size used.

Anatomy of a numerical differentiation

It has already been mentioned about numerical difficulties that are encountered in numerical differentiation. Especially when the function to be differentiated is trigonometric and it has an oscillating behavior, the estimated value of the derivative may be away from the analytic true value, in spite of a sufficient precision ensured in the calculations. Reducing the step size is not always the answer to improve accuracy of the estimated results. Since in numerical differentiation a very small subtracted value is divided by another very small value, the round-off error becomes predominant when the step size is reduced beyond an optimum limit.

Let us calculate the first to fourth derivatives of the trigonometric function,

$$f(x) = x.\sin(x)$$

Its analytic solutions with $x = \pi/4$ are as follows:

$$f'(x) = x.\cos(x) + \sin(x) = 1.262467148$$

$$f''(x) = -x.\sin(x) + 2.\cos(x) = 0.858853195$$

$$f'''(x) = -x.\cos(x) - 3.\sin(x) = -2.676680711$$

$$f^{iv}(x) = x.\sin(x) - 4.\cos(x) = -2.273066758$$

All of the three methods, namely forward, backward and central difference with normal- and higher-order are used and the four derivatives, mentioned above, are computed with $x = \pi/4$. The Richardson extrapolation was used in the case of formulas of second- and higher-orders. The step size was varied from 0.1 to 0.001. Table 3.1.5 shows the numerical results for comparison among various methods.

Table 3.1.5. A Comparison of Results by Numerical Methods.
[Function : $f(x) = x.\sin(x)$]

Derivatives	FORWARD		BACKWARD		CENTRAL	
	Normal-order	Higher-order	Normal-order	Higher-order	Normal-order	Higher-order
<u>FIRST</u>						
h = 0.1	1.3008574	1.2623848	1.2151614	1.2625733	1.2624663	1.2624671
h = 0.01	1.2667167	1.2624671	1.2581284	1.2624672	1.2624671	1.2624671
h = 0.001	1.2628961	1.2624671	1.2620373	1.2624671	1.2624671	1.2624671
<u>SECOND</u>						
h = 0.1	0.57897864	0.85959065	1.1122721	0.85823770	0.85885294	0.85885320
h = 0.01	0.83195482	0.85885388	0.8854864	0.85885252	0.85885320	0.85885320
h = 0.001	0.85617519	0.85885320	0.8615285	0.85885319	0.85885320	0.85885320
<u>THIRD</u>						
h = 0.1	-2.9639433	-2.6754579	-2.2875396	-2.6784899	-2.6766773	-2.6766807
h = 0.01	-2.7102626	-2.6766792	-2.6420761	-2.6766823	-2.6766807	-2.6766807
h = 0.001	-2.6800849	-2.6767060	-2.6732659	-2.6767012	-2.6766780	-2.6766772
<u>FOURTH</u>						
h = 0.1	-1.3846516	-2.2782350	-3.0027299	-2.2692239	-2.2730652	-2.2730668
h = 0.01	-2.1904592	-2.2730706	-2.3540767	-2.2730630	-2.2730663	-2.2730662
h = 0.001	-2.2651880	-2.1510201	-2.2811752	-2.3546350	-2.2739218	-2.2639976

It is evident from Table 3.1.5 that, as the order of differentiation is increased, the numerical estimation deteriorates. In fact, the estimation is better with a larger step size than a smaller one especially when the fourth derivative is evaluated by higher-order methods. In the case of central difference and normal second-order method, the relative error in evaluating the fourth derivative was -0.03760% when a step size of 0.001 was used. The estimation was correct up to the sixth decimal place when a step size was 0.01. In the case of the fourth-order central difference method, the estimation was off by 0.40058%, whereas it was correct up to the seventh decimal place with a 0.1 step size. Since numerical formulas for the fourth-order have more functions for evaluations, the round-off error predominates for a very small step size.

It is also interesting to note that decreasing the step size from 0.1 to 0.001 does not affect precision up to the seventh decimal place when the fourth-order central difference scheme was used to evaluate first, second and third derivatives of the function considered. It should be noted at this point that the precision of different numerical schemes for differentiation also largely depends on the type and behavior of the function to be differentiated. There are certain trigonometric functions for which none of the numerical methods works satisfactorily.

References

1. **Burden, R. L. and Faires, J. D.**, *Numerical Analysis*. Fifth edition, Prindle, Weber & Schmidt, Boston, 1993.
2. **Dorn, W. S. and McCracken, D. D.**, *Numerical Methods with FORTRAN IV Case Studies*. John Wiley & Sons, New York, 1972.
3. **Hamming, R.W.**, *Numerical Methods for Scientists and Engineers*. Second Edition, International Student Edition, McGraw-Hill, Tokyo, Japan, 1973.
4. **Hildebrand, F. B.**, *Introduction to Numerical Analysis*. Second Edition, Dover Pub, New York, 1987.
5. **Hornbeck, R. W.**, *Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ, 310 pp., 1975.
6. **Kreyszig, E.**, *Advanced Engineering Mathematics*. Seventh edition, John Wiley & Sons, New York, 1993.
7. **Wylie, C. R., Jr. and L. C. Barrett**, *Advanced Engineering Mathematics*. Fifth edition, McGraw-Hill Book Co., New York, 1982.

CHAPTER 3.2

Real Roots of Algebraic Equations

The solution of a single equation for its one and only independent variable produces its root or roots. The roots are often called the zeros of the equation, because when they are substituted for the independent variable in the equation, it returns a zero value. The following quadratic equation

$$a.x^2 + b.x = c$$

can be written in such way that all the terms are collected on one side of the equal sign and the other side becomes zero, that is,

$$a.x^2 + b.x - c = 0$$

Since it is a quadratic equation, there will be two values of x that will satisfy it. In other words, there will be two roots or two zeros of the above equation when the coefficient a is not equal to zero. Depending upon the values of the coefficients a , b and c these roots may be real or complex/imaginary. In our discussion, only real roots will be considered. The methods developed will be largely appropriate for polynomial equations, but, at the same time they can be used to find roots of transcendental equations involving trigonometric, exponential, and other nonalgebraic functions.

It is very important to understand that our concern is to find an approximate root rather than an exact one. The necessity of finding real roots of a continuous function is common place in engineering where absolute accuracy is seldom demanded. Finding real roots of an algebraic equation implies that the positive and the negative terms of the function almost cancel each other, thus round off error becoming inevitable.

It is not practically possible to locate the roots precisely just by looking at the equation. One ancient method of tracing roots is to draw the function against the independent variable and note where it crosses the axis of the independent variable. Unfortunately this graphical method is not suitable for easy implementation on digital machines. Rather the idea that the function should return values of opposite signs on either side of the real root can be very helpful in roughly locating the same using the number crunching machines. Later a subroutine **SEARCH** will be developed on this principle.

When an equation bears more than one roots having the same value, it is said to have a multiple root. Let us consider the following cubic equation in **x**,

$$\begin{aligned} f(x) &= x^3 - 3a.x^2 + 3a^2.x - a^3 \\ &= (x - a)^3 \end{aligned}$$

Since it is a cubic equation in **x**, when $f(x) = 0$, there should be three zeros of it. After the equation is written as $f(x) = (x - a)^3$, it becomes obvious that **x** can have just one value, which is **a** but with **multiplicity 3**, for which $f(x)$ becomes zero. In such a case it is said that the equation has a multiple root equal to **a** with multiplicity **3**.

In the case of a multiple root, the function may not change sign on the two sides of the root, and it becomes a special case where the common subroutine **SEARCH** does not work. The problem of such multiple zeros can pose trouble. One way to cope with this situation is to look for a sign change in the function as well as in the first derivative of the function. In the case of the former, the sign change indicates all odd-order zeros whereas in the case of the latter it indicates all even-order zeros. Thus, as it seems, the task of isolating multiple root is not that straightforward, and, in all practical situations, the value itself is important, and not how many roots exist with the same value. Among the methods discussed, the Modified Newton's method seems suitable to find the value of the multiple root with minimum effort. With the exception of the Bisection and False Position methods, which do not work for a multiple root for which the function does not change sign on its two sides, all the rest of the three methods discussed do.

A few of the simpler methods of finding real roots, such as (i) Iterative, (ii) Newton-Raphson, (iii) Modified Newton, (iv) Secant, (v) Bisection, and (vi) False Position will be discussed here.

Iterative method

In this method, the function is rearranged in such a way that the single power of the independent variable is made equal to an expression on its right hand side. Now, the independent variable on the right hand side of the equation being taken as known (a guess is made for it), the left side value is

evaluated. This value is again substituted on the right hand side and a new value is calculated. This iterative process is continued, and, at each iteration, two consecutive values of the independent variable are compared. When the absolute difference between them falls below a predetermined small value, the most recent value of the independent variable is considered as the root of the equation.

In Part 2, while explaining arithmetic statement function, this method was explained in example program 2.13. Instead of repetition, it can be said that this iterative method is not very efficient and that the root that is going to be found by this method greatly depends on how the rearrangement of the original function has been made. As an example, if the quadratic equation shown above is arranged as

$$f(x) = x_{i+1} = -\frac{a}{b} \cdot x_i^2 - \frac{c}{b}$$

the iterative method can calculate only one root, whereas if the quadratic equation is rearranged as

$$f(x) = x_{i+1} = \sqrt{\frac{-b \cdot x_i - c}{a}}$$

for appropriate values of **a**, **b** and **c**, there will be two roots that can be calculated by the iterative method.

Newton-Raphson method

This is the classical method for root finding, although it has some obvious limitations. In this method, the local tangent value of the function is substituted for the function, and the zero of this line is taken as the next approximation of the zero of the equation. This process is repeated until the absolute difference between two consecutive approximations of the root becomes close to zero when it is expected that the root has been found.

If a point x_0 is considered that is not a root of the function $f(x)$ but is reasonably close to it, the function can be expanded by the Taylor Series about x_0 to give

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!} \cdot f''(x_0) + \dots$$

According to the definition of a root, if $f(x)$ is set to zero, x must be a root. Taking only the first two terms of the above Taylor Series,

$$f(x_0) + (x - x_0).f'(x_0) = 0 \quad 3.2.1$$

In fact, Eq. (3.2.1) is the equation of the tangent line at x_0 . Now, solving it for $x = x_1$, the next approximation is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

The general expression for the above, known as the expression for Newton-Raphson method becomes,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad 3.2.2$$

The conditions for convergence of this method are that:

- (i) x_0 is sufficiently close to a root of $f(x) = 0$, and
- (ii) $f'(x)$ is not too close to zero.

One of the limitations of the Newton-Raphson method is that it requires analytical expression for the first derivative of the function. This may be difficult to obtain or may not be available at all for a complicated function. On the other hand, when the function is not explicit, which is the case with boundary value problems discussed in Chapter 3.6, this method with all its virtues cannot be used. Another limitation is the inflection point where, due to the nature of the function, which is usually oscillatory as shown in Figure 3.2.1, the scheme iterates infinitely between two consecutively calculated approximations.

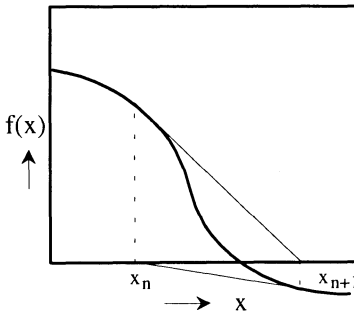


Figure 3.2.1. Inflection point

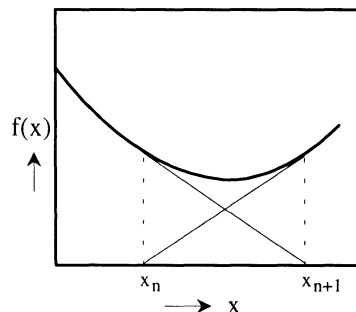


Figure 3.2.2. Local minimum

In the case of a multiple root, where the function is a tangent to the x -axis, this method is very slow near the root and also the first derivative of the

function may become too small and cause trouble in the division step giving a very large difference between the calculated values of two consecutive approximations.

The local minimum of a function, as shown by Figure 3.2.2, is not easily perceived. In this case, the sign of the function does not change, unlike the case of a turning (inflection) point, but, in any event the infinite loop causes trouble in a machine.

A search for the region where the zeros lie

In order to start any numerical method for root finding, it is necessary to provide one or two guesses of the root, which should be reasonably close to actual value(s). In order to do this, the subroutine **SEARCH** is developed. A limit on the roots and a searching step being provided, it calculates the values of the given function and identifies two points on either side of the root. If the product of $f(x)$ and $f(x+h)$ is negative, where h is the search step (preferably a positive fraction), the value of the root should be between x and $x+h$. If the product is positive throughout the interval with a sufficiently small value of h , either there is no root in the interval, or there may be a multiple root where the function is a tangent on the x -axis without crossing it.

When such a thing happens, the user gets an option of changing the interval or the step or both and going for any of the three methods (Newton-Raphson, Modified Newton, and Secant) to try to find the root or simply quit. If, by chance, the Bisection or False Position method is chosen, requiring two guesses on either side of the real root, it gives a message that since the function does not change sign in the interval, either of these methods does not work, and the control is transferred to the main menu. It has a safe exit when the value of $f(x)$ or $f(x+h)$ becomes too large beyond the capacity of a digital computer. The listing of this subroutine is provided in Appendix A.3.2.1.

Modified Newton method

This method was specially developed to overcome the slow approach of the Newton-Raphson method close to a multiple zero of a smooth function.

If a new function $u(x)$ is defined as

$$u(x) = \frac{f(x)}{f'(x)} \quad 3.2.3$$

so that it bears the same roots as the original function $f(x)$ does, meaning that $u(x)$ can take the place of $f(x)$ without changing the definition of the root finding problem. This is because $u(x)$ is zero everywhere when $f(x)$ is zero.

Now, $u(x)$ is differentiated with respect to x and using the chain rule, we obtain

$$\begin{aligned}\frac{du(x)}{dx} &= u'(x) = \frac{d}{dx} [f(x) \cdot \{f'(x)\}^{-1}] \\ &= -f(x) \cdot [f'(x)]^{-2} \cdot f''(x) + f'(x) \cdot \frac{1}{f'(x)} \\ &= 1 - \frac{f(x) \cdot f''(x)}{[f'(x)]^2}\end{aligned}\quad 3.2.4$$

The general recurrence formula can be written pursuing the Newton-Raphson logic as

$$x_{n+1} = x_n - \frac{u(x_n)}{u'(x_n)} \quad 3.2.5$$

where, $u'(x_n)$ is given by Eq. (3.2.4).

This method still suffers the first limitation of the Newton-Raphson method, and, in addition, it also requires the analytical expression for the second derivative of the function. Though this method offers the same convergence rate as that of Newton-Raphson, it is more time consuming because of an extra calculation required for $f''(x)$ to be evaluated at each iteration. Besides being somewhat more efficient in evaluating the multiple zero of a smooth function, this method does not offer any other advantage.

Secant method

Both the Newton-Raphson and Modified Newton methods require the analytical expression of the first derivative of the function and hence are useless for finding the root of an implicit function. In fact, the Secant method is based on the same logic as the Newton-Raphson method, where the first derivative of the function is expressed by a forward difference formula discussed in Chapter 3.1. Thus the Secant formula becomes,

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{f(x_{n+1}) - f(x_n)}{(x_{n+1} - x_n)}} \quad 3.2.6$$

From the Eq. (3.2.6), it is obvious that, to start the scheme, two initial guesses of the root, namely x_0 and x_1 (when $n = 0$) are to be supplied. If the difference between these two is denoted by D , Eq. (3.2.6) can be rewritten after rearranging as

$$x_{n+1} - x_n = D = - \frac{f(x_n) \cdot D}{f(x_{n+1}) - f(x_n)} \quad 3.2.7$$

For all practical purposes, this method is more advantageous than the proper Newton-Raphson, although it may require a few more iterations for convergence. A classic use of the Secant method is in solving boundary value problems discussed in Chapter 3.6.

It is important to decide a suitable convergence criterion for all the methods already discussed or going to be discussed. The choice of such a criterion decides the number of iteration and precision in the obtained value of the root.

Convergence criteria

A typical stopping criterion is to check the absolute difference between two successive approximations against a predetermined small quantity to the order of 10^{-6} . This does not say, however, that the approximation differs from the root by this amount. Nevertheless, the programs developed in this chapter use this criterion except for the Bisection method.

The second criterion is to check the absolute value of the function calculated at the approximation against a sufficiently small positive value. It should be realized that it may not be possible to achieve both the criteria at the same time.

There is a third criterion that is a kind of special case for the Bisection and the False Position methods. Since, in both of these methods, the product of the function values at either side of the root is computed to decide which way the iteration should proceed, this product is taken as a convergence criterion. When it is less than or equal to the error bound, the iteration stops.

Bisection method

This method is suitable for a real root for which the function changes sign from one side of the function to the other. For such a function, the Bisection method can locate the root with any desired accuracy. Let us consider a function that has a distinct real root in the interval $a < x < b$.

It is a requirement of this method that two initial guesses should be on either side of the root. Thus the product of $f(a)$ and $f(b)$ should be negative to start with. Now the interval between a and b is halved, and the new location is called x_m , so that

$$x_m = \frac{a + b}{2} \quad 3.2.8$$

There may be two cases,

- (i) $f(b)$ is negative, meaning that the root is between x_m and **b** ($x_m < x < b$), and
- (ii) $f(b)$ is positive. In this case, $f(x)$ has not crossed the axis between x_m and **b**, that is, the root is between **a** and x_m ($a < x < x_m$).

In the case of (i), the interval that contains the root, i.e., x_m and **b**, is halved calling x_m as **a** and continuing the process with Eq. (3.2.8) until the absolute value of the function at the latest x_m or the value of the product $f(x_m).f(b)$ becomes less than the error bound selected in the beginning. Then x_m is taken as the value of the root.

In the case of (ii), the interval that contains the root, i.e., **a** and x_m , is halved calling x_m as **b** and continuing the process with Eq. (3.2.8) until the absolute value of the function at the latest x_m or $f(x_m).f(b)$ becomes less than the error bound.

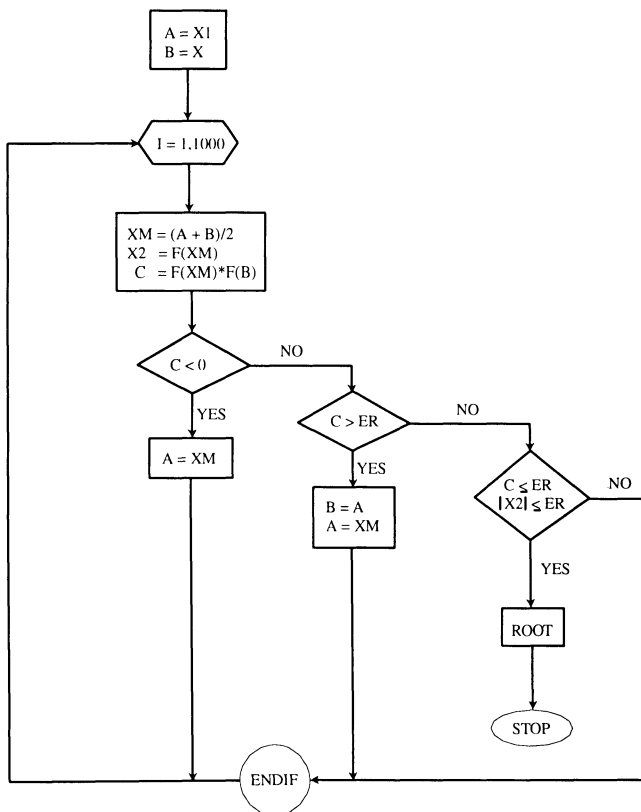


Figure 3.2.3. Flow diagram of Bisection algorithm.

The flow chart of the algorithm of this method in Figure 3.2.3 will help in the understanding of the algorithm's functioning. The program listings of all the five methods discussed here are given in Appendix A.3.2.2 (in the listing of the subroutine **ROOT**).

One limitation of this method is that it cannot locate a multiple root of a smooth function for which the function touches the x-axis without crossing it, thus not allowing the function to change sign on the two sides of the root.

False Position method

This method, better known as **Regula Falsi**, is an ancient one but does a better job than the Bisection method which is very slow near the root. Like the Bisection method, it also requires two initial guesses on the two sides of the roots with opposite signs, but, instead of halving the interval, it uses the straight line passing through these guesses. Though it is faster than the Bisection method in the close vicinity of the root, its one-sided slow approach to the zero restricts its use on a digital machine.

Let us use the following diagram (Figure 3.2.4a) for the derivation of an expression for this method. Later, Figure 3.2.4b will be used to develop an improvement on it to get rid of its worst bottleneck — its one-sided approach. The improved version has been included in the subroutine **ROOT**, calling it again the False Position method.

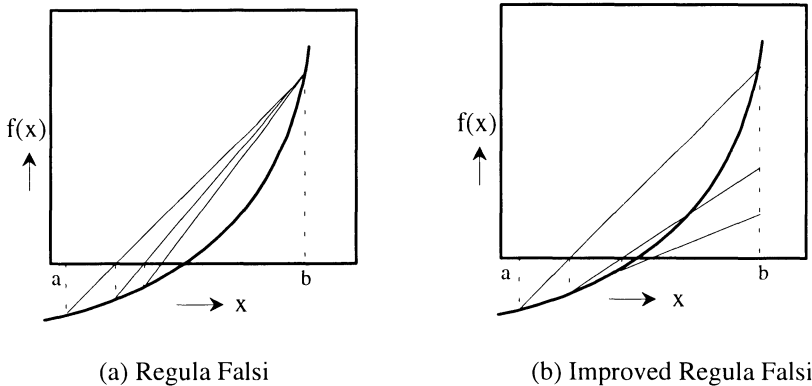


Figure 3.2.4. The working of **Regula Falsi** and its improved version.

If **a** and **b** are the two initial guesses of the root as shown in Figure 3.2.4, the equation of straight line passing through them is given by

$$f(x) = f(a) + \frac{f(b) - f(a)}{b - a}(x - a) \quad 3.2.9$$

Now, x becomes the root of this equation when $f(x)$ is zero. Solving for x , Eq. (3.2.9) gives,

$$x = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)} \quad 3.2.10$$

Once the method is started, after the first step, either a or b is kept unchanged, and the other is allowed to assume the calculated value of x , until an appropriate convergence criterion is met. In Figure 3.2.4, b is kept unchanged while a approaches the root from one side.

False Position method (improved)

The improvement in this case is to let both a and b of Eq. (3.2.10) approach the root simultaneously, thus accelerating the convergence. It is done by dividing the function value by 2 during each iteration. The function, which should be of a or of b , is decided by the sign of the product of $f(a)$ and $f(x)$. Thus when this product is negative, x becomes a new b and $f(a)$ is divided by 2; if this product is positive, x becomes a new a and $f(b)$ is divided by 2. When the absolute value of the product is less than a given small error bound, the convergence is believed to have been reached and the root is taken as x .

In most of cases this method is faster than the Bisection method but suffers the limitation that number of iterations for convergence can not be predicted. Its convergence and the flow diagram are shown in Figures 3.2.4b and 3.2.5, respectively.

Development of a program to compute a root in a given interval

The program developed has been listed in Appendix A.3.2.4 with the name **MAIN.FOR**. When the compiled program is executed, through a main option menu, it offers an option for any of the five methods, namely, Newton-Raphson, Modified Newton, Secant, Bisection and False Position. Once opted, it asks for a search interval, search step and tolerance. It uses the subroutine **SEARCH** to roughly locate the interval where the root lies which is listed in Appendix A.3.2.1.

In case the function does not change sign during the given interval, a message tells the user that there may be a multiple root or no root in this interval, and the user can go back and change the interval or the step size or call the subroutine **ROOT** to find the root using any of the iterative, the Newton-Raphson, or the Modified Newton method. The subroutine **ROOT** has been developed with all the five methods discussed and is listed in

Appendix A.3.2.2 as **ROOT.FOR**. Even if no root is found, the user is informed about it, and the control goes back to the main option menu. If, by chance, the Bisection or the False Position method is selected, it tells about the impossibility of using either of them, because the function has not changed sign in the specified interval.

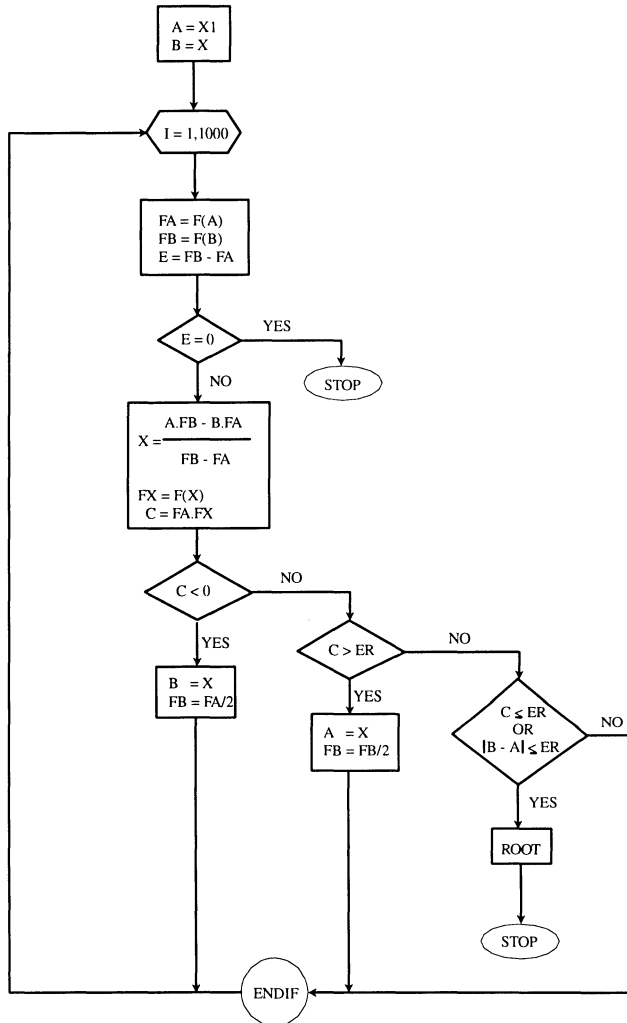


Figure 3.2.5. Flow diagram of improved False Position.

If the root is found by the subroutine **SEARCH** itself, the program informs the user about it and gives a hint that it may be a multiple root. The stopping criteria for **SEARCH** are (i) the given interval is covered, and (ii)

the function changes sign and (iii) root is found, that is, the corresponding absolute value of the function is less than or equal to the error bound, which has been provided as 10^{-6} for all the methods.

The function and its derivatives are provided through an external program called **FCTN.FOR**. This program should be written separately by the user and given the same name **FCTN.FOR**. An example is shown in Appendix A.3.2.3 using a sixth-order polynomial.

The results are shown on the screen and are also written and saved in a file called **DATA.OUT**, which is created by the program during its execution. The results include the value of the root, corresponding function value, and number of iterations for convergence.

Development of a program to compute all the roots in a given interval

This is a modification of the previous program offering the advantage of being able to check for all the possible roots in a given interval. Its operation is exactly like **MAIN FOR** and is listed in Appendix A.3.2.5 as **ZEROS.FOR**. It also uses the subroutines **SEARCH** and **ROOT** and the external function program **FCTN.FOR**.

Even if no root is found by **ROOT**, the user is informed about it and also about the search interval and is asked if he or she is interested in checking the rest of the interval or wants to go back to the main option menu.

In both the programs, it is necessary for the user to define the first and the second derivatives of the function as well as the function itself. In the case where the derivatives are not known, the Newton-Raphson and Modified Newton methods cannot be used. Nevertheless, **FD** (the first derivative) and **FDD** (the second derivative) should be defined in the external program **FCTN.FOR**. Only it should be remembered, however, that these are phantom functions and that Newton-Raphson or Modified Newton method must not be used to find root(s) of the function **F(X)**. Unless these functions are present in **FCTN.FOR**, there will be a compilation error. These functions may be defined in the following manner, or in any other suitable manner:

```
FUNCTION FD(X)
  FD = X
  RETURN
END
```

and

```
FUNCTION FDD(X)
  FDD = X
  RETURN
END
```

Table 3.2.1. Roots calculated by different methods.

Function	Search interval/ Step	R O O T F I N D I N G M E T H O D S									
		Newton-Raphson		Modified Newton		Secant		Bisection		False Position	
		Root value	Itms.	Root value	Itms.	Root value	Itms.	Root value	Itms.	Root value	Itms.
$f_1(X)$	-10, 10/0.1	-8.000000	2	-8.000000	2	-8.000000	2	-8.000000	16	-8.000000	1
		1.000000	2	1.000000	2	1.000000	3	0.999999	16	1.000000	2
		3.000000	2	3.000000	2	3.000000	3	3.000000	17	3.000000	3
		5.000000	2	5.000000	2	5.000000	2	5.000000	3	5.000000	10
		-7.348469	3	-7.348469	3	-7.348469	4	-7.348472	13	-7.348469	4
$f_7(X)$	-10, 10/0.1 5, 6/0.1	7.348469	6	7.348472	3	7.348471	4	7.348467	10*	7.348470	2
		4.300738	37	4.299043	19	@	-	-	-	-	-
$f_2(X)$	-10, 10/0.1 2, 3/0.1	-0.756945	3	-0.756945	3	-0.756945	3	-0.756639	8	-0.756945	6
		2.719231	10	2.718064	3	@	-	-	-	-	-

*Root was found by reducing step size to 0.01 or by changing the search interval to 5 to 10.

@Secant method was found to be unable to calculate multiple roots.

Discussion of results

The programs **MAIN.FOR** and **ZEROS.FOR** have been run with various types of functions, ranging from a sixth-order polynomial, polynomial with multiple root where the function is a tangent to x-axis, to a complicated exponential that has a spurious solution. It has been observed in general that for real and normal (not multiple) roots, all the methods work with variable accuracy and efficiency (number of iterations and precision) and that the type of function and the initial guess or guesses are very important factors in the functioning of the methods. The results are shown in Table 3.2.1.

Sometimes, with a coarse step size, the root may not be found, whereas with a finer step size, it produces the root. The coarse or fine step size essentially provides different guesses to start a method; these guesses are calculated by the subroutine **SEARCH**.

For all the functions tested, False Position required fewer iterations than Bisection to produce the same or even higher precision. In most of the cases, a finer step size yielded a root with higher precision than with a coarse step size, though it has not been always true.

Following are some examples of various functions that were used to find root(s) by either of the programs developed here (**MAIN.FOR** or **ZEROS.FOR**):

Sixth-order polynomial

$$f_1(x) = x^6 + x^5 - 49.x^4 + 69.x^3 + 120.x^2 + 98.x - 240$$

Roots : -8.0, 1.0, 3.0 and 5.0 (other two roots are complex)

Polynomial with multiple root

$$f_2(x) = x^4 - 8.6.x^3 - 35.51.x^2 + 464.4.x - 998.46$$

Multiple root = 4.3

Other roots : -7.34847 and 7.34847

Transcendental equation with spurious solution

$$f_3(x) = e^{a.x} - x^2$$

For $x = 0$, spurious solution;

For $a = 0.73575888$, roots : -0.75695 and 2.71828 (multiple)

These analytical/exact answers are not tabulated in Table 3.2.1 due to space considerations; nevertheless, readers may want to analyze the relative error in this case.

When a multiple root is suspected, the search interval should be given sufficiently small. In the course of finding other real roots, if a wide interval is specified, it may so happen that, after calculating one root, the program gives a message to its user to go ahead for finding the multiple root in the rest of the interval. Now, the subroutine **SEARCH** may return the initial search value at the end of the interval specified, which is actually entered as the initial guess for the next search. Depending upon the function, this initial guess may be far away from the multiple root. Thus the program may miss the multiple root, and it may revolve about the other real root. Thus if a small search interval is chosen, and, if there is a multiple root for which the function does not change sign on the two sides of it, it is very likely that the root will be found by the program.

References

1. **Burden, R. L., and Faires, J. D.**, *Numerical Analysis*. Fifth edition, Prindle, Weber & Schmidt, Boston, 1993.
2. **Carnahan, B., Luther, H. A., and Wilkes, J. O.**, *Applied Numerical Methods*. John Wiley & Sons, New York, 1969.
3. **Dorn, W. S. and McCracken, D. D.**, *Numerical Methods with FORTRAN IV Case Studies*. John Wiley & Sons, New York, 1972.
4. **Hamming, R.W.**, *Numerical Methods for Scientists and Engineers*. Second Edition, International Student Edition, McGraw-Hill, Inc., Tokyo, Japan, 1973.
5. **Hildebrand, F. B.**, *Introduction to Numerical Analysis*. Second Edition, Dover Pub, New York, 1987.
6. **Hornbeck, R. W.**, *Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ, 310 pp., 1975.
7. **Kreyszig, E.**, *Advanced Engineering Mathematics*. Seventh edition, John Wiley & Sons, New York, 1993.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

CHAPTER 3.3

Numerical Integration

Integration is the inverse operation of differentiation. Subtractions followed by divisions in differentiation are counteracted by multiplications followed by additions in integration. Integration of a function over a given limit is equivalent to finding the area of the functional curve between the limit and the axis of the independent variable. Thus the integral I , given by

$$I = \int_a^b f(x) \cdot dx \quad 3.3.1$$

is the shaded area shown in Figure 3.3.1.

In Eq. (3.3.1), I is called the definite integral, $f(x)$ the integrand, a the lower limit, and b the higher limit of the same the values of which should be known. In numerical integration the function $f(x)$ may be analytic within the lower and upper limits a and b , respectively or may be available at discrete points. One or both of the limits may sometime be infinite, and, when this occurs, the integral is called an improper integral of the first kind. The improper integral of the second kind is that for which the integrand is singular at either or both the limits of integration, though it has a finite value. There are ways to handle such situations numerically.

In the case of differentiation, it has been observed that the value of the function to be differentiated is made at discrete points, and, thus, the approximating formula cannot anticipate the behavior of the function between these points. Numerical integration, on the other hand, is more stable and certain.

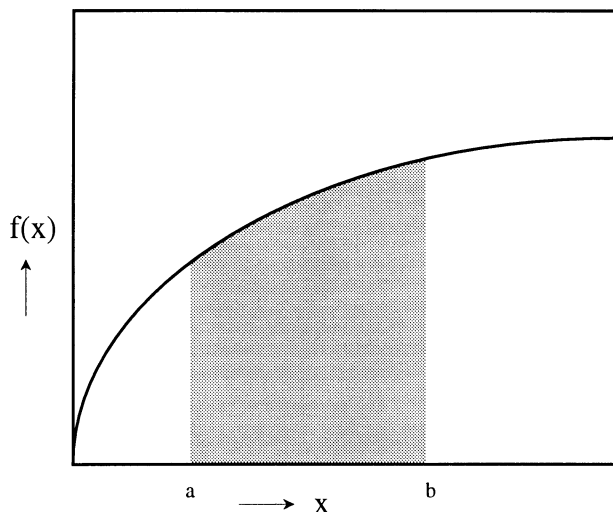


Figure 3.3.1. Graphical representation of definite integral.

Analytical evaluation of a definite integral is often very difficult. Many times closed form analytical expressions for the integrals of certain simple looking functions may not exist. In such cases, we resort to integrating the functions numerically. It should be remembered that the closed form analytical expressions of definite integrals has many advantages and should be used where possible. However, when a function is available at discrete points, numerical evaluation of its integral becomes necessary.

Generally, the formulas of numerical integration evaluate the function to be integrated at the two extreme limits or end points. This sometimes causes a problem, especially when the function gets too large a value (infinite with respect to the capacity of a particular computer) at either or both the limits, or when one of the limits itself is too large or infinite. The Gaussian type formulas come in handy in such cases of improper integrals. In the present context only the Trapezoidal, Simpson's, Gauss-Legendre Quadrature, and Romberg methods will be discussed in detail.

The word quadrature is, in fact, a synonym for numerical integration but has been used to express only the Gaussian type formulas where unequally spaced base points are employed.

Trapezoidal rule

Let us consider the integral denoted by the area **abcd** in Figure 3.3.2. This area can be approximated considering **abcd** to be a trapezoid by drawing a straight line **cd** as shown.

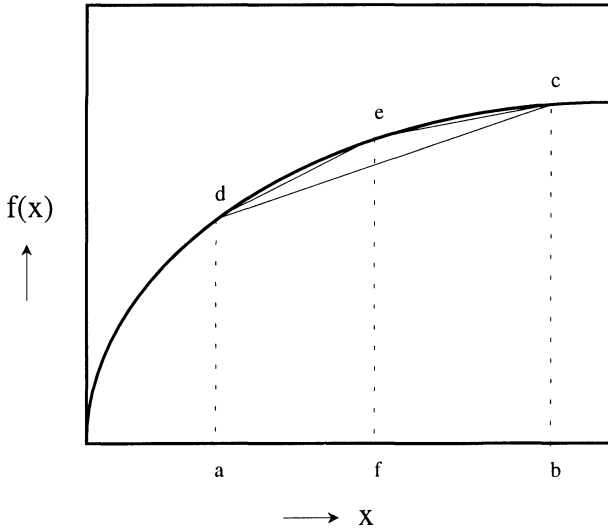


Figure 3.3.2. Approximation of an integral by trapezoids.

The area I of the trapezoid **abcd** in Figure 3.3.2 is

$$I = \frac{f(a) + f(b)}{2} (b - a) = \frac{h}{2} [f(a) + f(b)] \quad 3.3.2$$

where h is the interval, $(b - a)$.

If the interval $(b - a)$ now is divided into two equal parts, there will be two trapezoids, the joint areas of which will again approximate the area under the curve or the integral. This time, however, the shorter straight lines **ce** and **ed** are closer to the curve than the line **cd**, showing a better approximation of the integral. The sum of the areas **afed** and **bfec** is

$$\begin{aligned} I &= \frac{f(a) + f\left(a + \frac{b-a}{2}\right)}{2} + \frac{f\left(a + \frac{b-a}{2}\right) + f(b)}{2} \left(\frac{b-a}{2}\right) \\ &= \frac{h}{2} [f(a) + 2f(a+h) + f(b)] \end{aligned} \quad 3.3.3$$

where, again, h is the new interval, that is, $(b - a)/2$.

Now, the new interval is divided into two equal intervals, there will be four trapezoids, and the approximation is going to be even better than the two previous ones, where the integral, I is given by

$$= \frac{h}{2} [f(a) + 2f(a+h) + 2f(a+2h) + 2f(a+3h) + f(b)] \quad 3.3.4$$

In this case, the interval h is equal to $(b - a)/4$

Thus examining Eqs. (3.3.2) through (3.3.4), a general expression for the integral, I , can be established as

$$I = \frac{h}{2} \left[f(a) + 2 \sum_{i=1}^{n-1} f(a + i \cdot h) + f(b) \right] \quad 3.3.5$$

where n is the number of equal divisions or panels, and h is equal to $(b - a)/n$.

The expression given by Eq. (3.3.5) is called the **Trapezoidal rule** for the simple reason that it was derived by splitting up the area into a number of trapezoids. As the number of panels is increased, apparently the approximation of the integral approaches closer to the exact value, though, in fact the round-off error, which is proportional to h for small value of h , may not allow it to happen.

The above derivation is simple, but it does not give any hint about the accuracy of the expression. In order to have a quantitative idea about the truncation error of this method, the Taylor Series expansion is used and finally we obtain,

$$I = \frac{h}{2} \left[f(a) + 2 \sum_{i=1}^{n-1} f(a + i \cdot h) + f(b) \right] - \frac{h^2}{2} (b - a) f''(\xi) \quad 3.3.6$$

where ξ lies between a and b . Thus the Trapezoidal rule is a second-order method. Now, $f''(\xi)$ can be estimated as

$$f''(\xi) = \frac{[f'(b) - f'(a)]}{b - a}$$

When the above expression is substituted for $f''(\xi)$ in Eq. (3.3.6), the Trapezoidal rule with end correction becomes a fourth-order method and is given by

$$I = \frac{h}{2} \left[f(a) + 2 \sum_{i=1}^{n-1} f(a + i \cdot h) + f(b) \right] - \frac{h^2}{12} [f'(b) - f'(a)] \quad 3.3.7$$

Though the form of the Trapezoidal rule given by Eq. (3.3.7) is of higher-order than the one given by Eq. (3.3.5), the latter is mostly used. When the function to be integrated can easily be differentiated analytically,

exact values of $f'(a)$ and $f'(b)$ can be had, and the expression given by Eq. (3.3.7) is preferable. If numerical representations are used for these first derivatives, the final error becomes dependent on the finite difference schemes used, which have been discussed in Chapter 3.1. Forward difference representations have been used in the programs developed here to evaluate $f'(a)$ and $f'(b)$.

Simpson's rule

The integrand was approximated by straight lines in the Trapezoidal method, whereas Simpson's rule does it by parabolic curves. This is the most widely used method in numerical integration and, in principle, is the same as the Trapezoidal method, the only difference being that, instead of approximating the area in small intervals by trapezoids, it is done by parabolas. The Trapezoidal method should be exact for first-degree polynomials, whereas Simpson's rule is exact for second-degree polynomials and lower. It has been observed that Simpson's rule gives an exact answer up to the third-degree without much extra effort than the Trapezoidal method. This is the reason for its universal acceptance.

The geometric derivation of this rule is simple and comes from the Richardson extrapolation formula derived in the context of numerical differentiation in Chapter 3.1. This formula can be written in the following form:

$$I(x) = I_2(x) + \frac{I_2(x) - I_1(x)}{\left(\frac{h_1}{h_2}\right)^r - 1} \quad 3.3.8$$

Now, considering that $I_1(x)$ and $I_2(x)$ are evaluated with step sizes h_1 and h_2 , respectively, where $h_1 = 2h_2$ and $r = 2$, by the Trapezoidal rule,

$$I_1(x) = \frac{h_1}{2} [f(a) + 2f(a+h_1) + 2f(a+2h_1) + \dots + f(b)] \quad 3.3.9$$

$$\text{and} \quad I_2(x) = \frac{h_2}{2} [f(a) + 2f(a+h_2) + 2f(a+2h_2) + \dots + f(b)] \quad 3.3.10$$

Substituting expressions from Eqs. (3.3.9) and (3.3.10) into Eq. (3.3.8) and placing h in the place of h_2 , after some algebraic manipulations we obtain,

$$I(x) = \frac{h}{3} [f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + 2f(a+4h) + \dots + 2f\{a+(n-2).h\} + 4f\{a+(n-1).h\} + f(b)]$$

Thus for n panels, the general expression of Simpson's rule can be written as

$$I = \frac{h}{3} \left[f(a) + 4 \sum_{\substack{i=1 \\ i \text{ odd}}}^{n-1} f(a + i \cdot h) + 2 \sum_{\substack{i=2 \\ i \text{ even}}}^{n-2} f(a + i \cdot h) + f(b) \right] \quad 3.3.11$$

From the final expression (3.3.11), to ensure that $(n-1)$ is odd and $(n-2)$ is even, it becomes obvious that the number of panels, n , for Simpson's rule should only be even.

Similar to the Trapezoidal rule, the above derivation is simple, but, again, it does not give the accuracy of the expression. In order to have a quantitative idea about the truncation error of this method, the Taylor Series approach is used finally to obtain

$$I = \frac{h}{3} \left[f(a) + 4 \sum_{\substack{i=1 \\ i \text{ odd}}}^{n-1} f(a + i \cdot h) + 2 \sum_{\substack{i=2 \\ i \text{ even}}}^{n-2} f(a + i \cdot h) + f(b) \right] - \frac{h^4}{180} (b - a) \cdot f^{iv}(\xi) + \dots$$

where ξ lies between a and b .

The error term has the fourth-order in h , and hence it is a fourth-order method. It is not practical to attempt approximating the fourth derivative in the error term. Instead, the derivatives of the function are assumed to be known at the end points of each double panel. Now the function can be approximated by a fourth-order polynomial over the two panels. The integral thus evaluated by summing up all the pairs of panels, is given by

$$I = \frac{h}{15} \left[7 \left\{ f(a) + 2 \sum_{\substack{i=2 \\ i \text{ even}}}^{n-2} f(a + i \cdot h) + f(b) \right\} + 16 \sum_{\substack{i=1 \\ i \text{ odd}}}^{n-1} f(a + i \cdot h) + h \{ f'(a) - f'(b) \} \right] \quad 3.3.12$$

The formula given by Eq. (3.3.12) is known as Simpson's rule with end correction and it gives sixth-order accuracy, though the final accuracy depends on the selection of the finite difference representations for the first derivatives $f'(a)$ and $f'(b)$. In this case, forward difference formulas also have been used in the program developed here.

Romberg method

This is a modification of the simple Trapezoidal method made by Richardson extrapolation to the point that it becomes even more efficient and accurate than Simpson's rule. Here, the Trapezoidal rule with a single panel is used to generate a preliminary approximation, and, in each step, the number of panels is doubled, and the Richardson extrapolation is subsequently used to improve the approximation until the fractional difference between two successive approximations of the integral becomes less than a small tolerance.

Thus $T_{i,1}$ ($i = 0$ to n) is used to denote the integrals calculated by the Trapezoidal rule, and $T_{i,j}$ ($j = 2$ to k) for the subsequent extrapolated values, where i denotes the number of times the initial interval is halved and j the subsequent number of iterations needed for convergence. It is the common experience that the number of columns is almost always substantially less than the number of rows to reach a comparable accuracy. In other words, the extrapolated values along the diagonal converge to the correct answer (within the tolerance given) much faster than the interval halved values obtained by Trapezoidal rule in the first column shown in the following table constructed to explain this process:

$T_{0,1}$						
$T_{1,1}$	$T_{0,2}$					
$T_{2,1}$	$T_{1,2}$	$T_{0,3}$				
$T_{3,1}$	$T_{2,2}$	$T_{1,3}$				
$T_{4,1}$	$T_{3,2}$	$T_{2,3}$				
$T_{5,1}$	$T_{4,2}$	$T_{3,3}$				
.....	$T_{5,2}$	$T_{4,3}$				
$T_{n-5,1}$		$T_{5,3}$				
$T_{n-4,1}$	$T_{n-5,2}$					
$T_{n-3,1}$	$T_{n-4,2}$	$T_{n-5,3}$				
$T_{n-2,1}$	$T_{n-3,2}$	$T_{n-4,3}$		$T_{n-k+1,k-2}$		
$T_{n-1,1}$	$T_{n-2,2}$	$T_{n-3,3}$		$T_{n-k+2,k-2}$	$T_{n-k+1,k-1}$	
$T_{n,1}$	$T_{n-1,2}$	$T_{n-2,3}$		$T_{n-k+3,k-2}$	$T_{n-k+2,k-1}$	$T_{n-k+1,k}$

Now, interval halving is done as follows:

$$T_{0,1} = \frac{h}{2} [f(a) + f(b)] \quad 3.3.13a$$

$$\begin{aligned} T_{1,1} &= \frac{h}{2} \left[f(a) + 2f\left(a + \frac{h}{2}\right) + f(b) \right] \\ &= \frac{T_{0,1}}{2} + \frac{h}{2} \cdot f\left(a + \frac{h}{2}\right) \end{aligned} \quad 3.3.13b$$

$$\begin{aligned}
 T_{2,1} &= \frac{h}{8} \cdot \left[f(a) + 2 \sum_{\substack{m=1 \\ m \text{ odd}}}^3 f\left(a + m \cdot \frac{h}{4}\right) + f(b) \right] \\
 &= \frac{T_{1,1}}{2} + \frac{h}{4} \sum_{\substack{m=1 \\ m \text{ odd}}}^3 f\left(a + m \cdot \frac{h}{4}\right)
 \end{aligned} \tag{3.3.13c}$$

$$\begin{aligned}
 T_{3,1} &= \frac{h}{16} \cdot \left[f(a) + 2 \sum_{\substack{m=1 \\ m \text{ odd}}}^7 f\left(a + m \cdot \frac{h}{8}\right) + f(b) \right] \\
 &= \frac{T_{2,1}}{2} + \frac{h}{8} \sum_{\substack{m=1 \\ m \text{ odd}}}^7 f\left(a + m \cdot \frac{h}{8}\right)
 \end{aligned} \tag{3.3.13d}$$

.....

$$T_{n,1} = \frac{T_{n-1,1}}{2} + \frac{h}{2^n} \sum_{\substack{m=1 \\ m \text{ odd}}}^{2^n-1} f\left(a + m \cdot \frac{h}{2^n}\right) \tag{3.3.13e}$$

By induction from Eqs. (3.3.13), the first column of the table is given by the general recursion equation:

$$T_{i,1} = \frac{T_{i-1,1}}{2} + \frac{h}{2^i} \sum_{\substack{m=1 \\ m \text{ odd}}}^{2^i-1} f\left(a + m \cdot \frac{h}{2^i}\right) \tag{3.3.14}$$

where **i** goes from **1** to **n**.

All the values in this column are calculated by the Trapezoidal rule. It is interesting to note that the function is not calculated more than once at the same point. Thus the function needs to be evaluated only $(2^n - 1)$ times for the summation series plus two times — once to calculate $f(a)$ and the second time to calculate $f(b)$, that is, a total of $(2^n + 1)$ times to compute the entire sequence.

From the second column onwards, the Richardson extrapolation technique is applied. If the second column is considered, by the Richardson extrapolation, $T_{n-1,2}$ is given by

$$T_{n-1,2} = T_{n,1} + \frac{T_{n,1} - T_{n-1,1}}{2^2 - 1}$$

$$= \frac{4T_{n,1} - T_{n-1,1}}{3} \quad 3.3.15a$$

Similarly,

$$\begin{aligned} T_{n-2,3} &= T_{n-1,2} + \frac{T_{n-1,2} - T_{n-2,2}}{2^4 - 1} \\ &= \frac{16T_{n-1,2} - T_{n-2,2}}{15} \end{aligned} \quad 3.3.15b$$

$$\begin{aligned} T_{n-3,4} &= T_{n-2,3} + \frac{T_{n-2,3} - T_{n-3,3}}{2^5 - 1} \\ &= \frac{64T_{n-2,3} - T_{n-3,3}}{63} \end{aligned} \quad 3.3.15c$$

$$\dots\dots\dots$$

$$T_{n-k+1,k} = \frac{4^{k-1} T_{n-k+2,k-1} - T_{n-k+1,k-1}}{4^{k-1} - 1} \quad 3.3.15d$$

By induction again from Eqs. (3.3.15), the recursion formula for extrapolated integral values is obtained as

$$T_{i,j} = \frac{4^{j-1} T_{i+1,j-1} - T_{i,j-1}}{4^{j-1} - 1} \quad 3.3.16$$

where **i** goes from **0** to **n** whereas **j** goes from **2** to **k**.

When Eq. (3.3.16) is examined, it is observed that for $i=0$ and $j=2$,

$$T_{0,2} = \frac{4T_{1,1} - T_{0,1}}{3}$$

Now, substituting the expressions for $T_{0,1}$ and $T_{1,1}$ from Eqs. (3.3.13a) and (3.3.13b), respectively in above, we obtain

$$T_{0,2} = \frac{h}{6} \left[f(a) + 4f\left(a + \frac{h}{2}\right) + f(b) \right] \quad 3.3.17$$

Eq. (3.3.17) is in fact the single application (two-panel) Simpson's rule. From this and examining Eq. (3.3.16) for other values of **i**, which is the number of times the interval is halved, it can be concluded that $T_{i,2}$ is the estimate of the integral obtained by the Simpson's rule with the number of applications equal to 2^i . So, $T_{n,2}$ is the integral by the Simpson's rule after halving the interval **h**, **n** times. A flow diagram of the Romberg integration method is given here (Figure 3.3.3), which will help in understanding the algorithm of this method.

The truncation error term for the second column is of fifth-order in h , the third column is of seventh-order in h , and so on. The criterion for convergence can be established in one of the two ways — (i) the absolute difference between the last two extrapolated values in the same column (same value of j), or (ii) the absolute value of relative error, expressed in fraction, between the last two extrapolated values in the same column is less than or equal to a given tolerance. In the program developed here, the relative error criterion has been used.

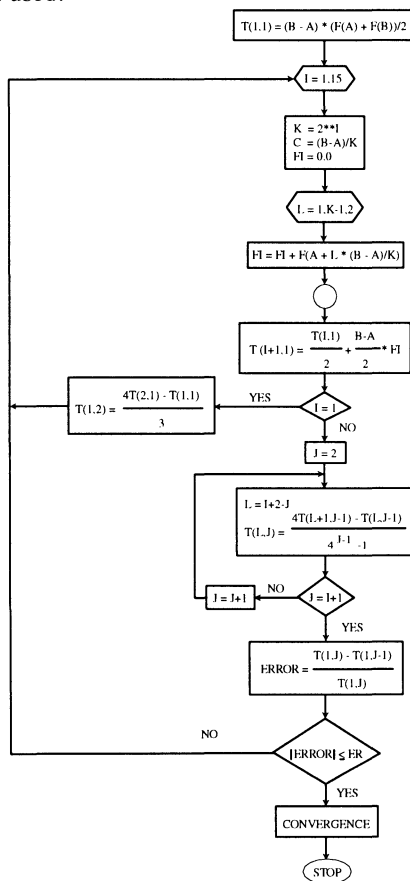


Figure 3.3.3. Flow diagram of the Romberg integration method.

One should be careful in deciding the tolerance value for terminating the Romberg integration procedure. If it is selected too small, the roundoff error may predominate, and the answer may deviate from the true value. On the other hand, it should not be too large. The effect of this tolerance value on the number of applications of the method is shown later.

All the three methods discussed so far use equally spaced area segments and the function to be integrated is evaluated at both the end points or the

integration limits. The Romberg method yields highly accurate results using fewer numbers of application of the formula (number of panels) than the Trapezoidal or Simpson's method. Due to its high efficiency and precision it is best suited to computers.

The following problem is chosen to show the power and accuracy of the Romberg integration method.

Example 3.3.1. Evaluate the following integral using the Romberg method:

$$I = \int_1^{10} \frac{1}{x} dx$$

Solution

The first two values are calculated by the Trapezoidal rule using Eq. (3.3.13a) and the recursion formula (3.3.14) are 4.95 and 3.293182, respectively. Then the extrapolated value is calculated by Eq. (3.3.16) as 2.740909. To calculate the next extrapolated value, the Trapezoidal rule is used again halving the interval to obtain it as 2.629221 and then the extrapolated value as 2.407901. After calculating a pair of extrapolated values, they are checked against a small tolerance. The convergence is believed to have been reached if the absolute value of the relative error between two calculated extrapolated values falls below the tolerance, if not the process continues. The Romberg table is shown below to give an idea of how the algorithm works:

Table 3.3.1. Values of integral and convergence in the Romberg method with a tolerance of 10^{-6} .

Trapezoidal rule		E x t r a p o l a t e d V a l u e s						
Panels	Values	$T_{i,2}$	$T_{i,3}$	$T_{i,4}$	$T_{i,5}$	$T_{i,6}$	$T_{i,7}$	$T_{i,8}$
1	4.950000							
2	3.293182	2.740909						
4	2.629221	2.407901	2.385700					
8	2.397737	2.320576	2.314754	2.313628				
16	2.327952	2.304691	2.303632	2.303455	2.303415			
32	2.309061	2.302763	2.302635	2.302619	2.302615	2.302615		
64	2.304213	2.302597	2.302586	2.302585	2.302585	2.302585	2.302585	
128	2.302986	2.302586	2.302585	2.302585	2.302585	2.302585	2.302585	2.302585

From the Table 3.3.1, it is observed that, even with 128 panels, the Trapezoidal rule produces a relative error of -0.017411%, whereas in the Romberg procedure, each column sequence converges to the true value of the integral. The convergence occurs with 64 panels if the tolerance is given as 10^{-4} without affecting the accuracy of the result.

Gauss-Legendre Quadrature

All the methods discussed so far suffer the drawback in handling singularities already mentioned, because they require evaluation of the integrand at both the limits of integration. Gauss-type quadrature formulas, in many occasions, beat this limitation. Orthogonal polynomials such as Jacobi, Legendre, Laguerre, Hermite, Chebyshev, etc., are very powerful in approximating various types of functions and are the basis of derivation of Gauss-type Quadrature formulas. Among the polynomials mentioned, the Legendre polynomial is widely used to derive the integration formulas known as Gauss-Legendre Quadrature formulas. In this arbitrarily spaced base points/intervals are used. The Legendre polynomials are as follows:

$$\begin{aligned}
 P_0(x) &= 1 \\
 P_1(x) &= x \\
 P_2(x) &= (3x^2 - 1)/2 \\
 P_3(x) &= (5x^3 - 3x)/2 \\
 P_4(x) &= (35x^4 - 30x^2 + 3)/8 \\
 &\dots\dots\dots \\
 P_n(x) &= \frac{2n-1}{n} \cdot x \cdot P_{n-1}(x) - \frac{n-1}{n} \cdot P_{n-2}(x)
 \end{aligned}$$

Derivations of Gauss-type formulas are complicated and will not be presented here. The formulas once derived are very simple but powerful and require less calculations. In this, the integral I is expressed as

$$I = \frac{b-a}{2} \left[w_1 \cdot f(x_1) + w_2 \cdot f(x_2) + w_3 \cdot f(x_3) + \dots + w_m \cdot f(x_m) \right]$$

where, $w_1, w_2, w_3, \dots, w_n$ are the weights to be given to the functional values $f(x)$, and **a** and **b** are the lower and upper limits of integration.

In this method, the integral is evaluated on an interval **z** between -1 and 1 in which the polynomials are orthogonal, instead of **x** between **a** and **b**. Thus, it becomes necessary to transform $-1 < z < 1$ to $a < x < b$. This is done by the following expression:

$$x_k = \frac{b+a}{2} + \frac{b-a}{2} \cdot z_k \quad 3.3.18$$

where, z_k are called roots or zeros of Legendre polynomials.

For m number of base points/panels, the general expression for Gauss-Legendre is thus given by:

$$I = \frac{b-a}{2} \sum_{k=1}^m w_k \cdot f(x_k) \quad 3.3.19$$

In order to evaluate an integral, the Gauss-Legendre Quadrature formula should be chosen with a certain number of base points. Stroud and Secrest (1966) listed values of roots and weights for from 2 to 256 base points. These values for from 2 to 10, 12, 15, 16, 20, and 24 base points/panels are listed in Appendix A.3.3.2 (Abramowitz and Stegun, 1972) as a data input file and also in Appendix A.3.3.6 in tabular form.

As observed from the values of zeros and weights listed, the number of zeros and weights for an even number of panels is exactly half the number of panels and for an odd number of panels, it is one unit more than half. The first root for an odd number of panels is always zero, all the rest have both plus and minus signs before them, just as all the roots for an even number of panels do. This makes the number of roots equal to the number of panels. Along with the roots, corresponding values of the weights also should be considered but without any change of sign in them. An example problem is solved to demonstrate the working of this method.

Example 3.3.2. Evaluate the following integral using three and four panels with the Gauss-Legendre Quadrature method:

$$I = \int_0^{\pi/2} \frac{x}{\cos^2 x} dx \quad [\text{exact answer : } 0.4388245]$$

Solution

First let us use three base points. For these, referring to Appendix A.3.3.6, the zeros and weights are as follows:

$$z_1 = 0.0, z_2 = 0.77459667 \text{ and } z_3 = -0.77459667$$

$w_1 = 0.88888889, w_2 = 0.55555556, w_3 = 0.55555556$, and the limits are $a = 0.0$ and $b = \pi/4 = 0.78539816$

Thus from Eq. (3.3.18),

$$x_1 = \frac{0.0 + 0.78539816}{2} + \frac{0.78539816 - 0.0}{2}(0.0) \\ = 0.3926990$$

Similarly,

$$x_2 = 0.3926990 + 0.3926990 * 0.77459667 \\ = 0.6968825$$

and

$$x_3 = 0.3926990 + 0.3926990 * (-0.77459667) \\ = 0.08851567$$

Therefore,

$$w_1 * f(x_1) = 0.88888889 * 0.3926990 / \cos^2(0.3926990) \\ = 0.4089561$$

$$w_2 * f(x_2) = 0.55555556 * 0.6968825 / \cos^2(0.6968825) \\ = 0.6583697$$

and

$$w_3 * f(x_3) = 0.55555556 * 0.08851567 / \cos^2(0.08851567) \\ = 0.04956269$$

Substituting these values into Eq. (3.3.19),

$$I = \frac{0.78539816 - 0.0}{2.0} * [0.4089561 + 0.6583697 + 0.0495627] \\ = 0.4386011 \quad [\text{exact answer : } 0.4388245]$$

Now, using four base points, the zeros and weights areas follows:

$$z_1 = 0.33998104, z_2 = 0.86113631, z_3 = -0.33998104, z_4 = -0.86113631 \\ w_1 = 0.65214515, w_2 = 0.34785484, w_3 = 0.65214515, w_4 = 0.34785484,$$

and x_1 through x_4 as well as $w_1 * f(x_1)$ through $w_4 * f(x_4)$ are calculated as,

$$x_1 = 0.5262094, x_2 = 0.7308666, x_3 = 0.2591888 \text{ and } x_4 = 0.05453163 \\ w_1 * f(x_1) = 0.4589387, w_2 * f(x_2) = 0.4585582, w_3 * f(x_3) = 0.1809125, \\ \text{and } w_4 * f(x_4) = 0.01902561$$

So, the value of the integral is calculated by Eq. (3.3.19) as

$$I = 0.39269908 * [0.4589387 + 0.4585582 + 0.1809125 + 0.0190256] \\ = 0.4388157 \quad [\text{exact answer : } 0.4388245]$$

Approximations to singularities

The applied mathematics that we have been dealing with so far have been finite. However, in integration, singularities are occasionally encountered, which are not so frequent in the engineering fields as they are in physics. Still this topic is important, and a brief discussion follows.

It has been discussed earlier in this chapter that singularities are of two types, one in which one limit of integration or both are infinite and the other where the value of the integrand becomes infinite at one of the integration limits. The former is known as an improper integral of the first kind and the latter as an improper integral of the second kind.

Improper integral of the first kind

When the upper limit is infinite, for the integral to have a finite value, it is necessary that the integrand should approach zero as the independent variable tends to be infinite. Thus if the integral is broken up into two parts, one with a large higher limit and the other with this higher limit value as its lower limit and infinity as its higher limit, the latter gets smaller as the higher limit of the former is increased. The first integral then can be solved by any suitable numerical method, and the second can totally be ignored. This certainly will introduce error, but an idea of this error can be had by continuing to change the upper limit of the first integral and comparing the successive results. When a further increment in the upper limit does not change the result considerably, it is likely that the second integral is really negligible. In other words, the answer is accepted when the absolute difference between two successive values of the first integral calculated lies within a small tolerance. An example will make this clear.

Example 3.3.3. Let us consider the following integral:

$$I = \int_0^{\infty} \frac{dx}{1+x^4} \quad [\text{exact answer} = 1.1107207]$$

Solution

If the upper limit is kept as **a** where **a** should have a large value, the original integral **I** can now be split into two integrals I_1 and I_2 as follows:

$$I = I_1 + I_2$$

where,

$$I_1 = \int_0^a \frac{dx}{1+x^4} \quad \text{and} \quad I_2 = \int_a^{\infty} \frac{dx}{1+x^4}$$

For different values of **a**, I_1 is evaluated by Simpson's rule with 20,000 panels and is given in tabular form below:

a	I_1
100.0	1.110698
500.0	1.110712
1000.0	1.110716
1500.0	1.110718
2000.0	1.110718

Thus it is seen that with a value of **a** between 1000.0 and 2000.0, the first integral is producing an answer correct up to five significant digits (a relative error of 2.462365×10^{-6} %). Using the Romberg method with a tolerance of 10^{-6} , and a value of **a** equal to 100.0, the first integral is calculated as 1.110717, which has the same precision as the above result obtained by Simpson's rule with 20,000 panels and a value of **a** equal to 1500.0.

Another way of handling such singularities is more analytical than numerical and may not be convenient for a person with an engineering background. It is done by finding a transformation that eliminates the singularity. It actually belongs to the domain of formal mathematical manipulation and outside that of numerical methods.

Improper integral of the second kind

These are improper integrals with finite limits that have an integrand that is singular at one or both the end points. Since, in all the methods discussed so far, with the exception of Gauss-Legendre Quadrature, it is necessary to evaluate the integrand at both the end points, they cannot be used directly to deal with such singularity. The Gauss Quadrature, on the other hand, should be used with caution. Sometimes the derivation of such formulas might include some part of the integrand, and, in this case, while the Gauss Quadrature method is used, this part should be eliminated. One example has been extracted from Hornbeck (1975) to explain this important point. The following integrand has a singularity at $x = 0$, because $\ln(x)$ cannot be evaluated at this point:

$$f(x) = \frac{\ln(x)}{1+x} \quad 3.3.20$$

Now, in the derivation of special Gauss Quadrature formulas of the above type, the term $\ln(x)$ has already been considered, and, hence, if this method (roots and weights of which are not given in this text) is to be used, the integrand should be without this term, that is, it should be

$$f(x) = \frac{1}{1+x}$$

and only two base points formula yields an answer, which is good up to four significant digits [exact solution is $-\pi/12$].

If the integral of the above function is to be evaluated between the limits 0 and 1 by the usual Gauss-Legendre Quadrature method using the integrand given by Eq. (3.3.20), it produces the value of the integral equal to -0.821414, when 24 base points are used, which is correct up to only two significant digits [exact answer is -0.822467033].

By the other methods that use both of these end points, the problem should be tackled in a manner similar to what has been done in the case of the singularity of the first kind or by interchanging a dependent variable with an independent one. The lower limit $a = 0$ in this case is causing a problem. Now, the Trapezoidal method, Simpson's rule, or Romberg integration method can be applied using a very small value of a , which, in reality, represents zero. If the value of a is varied, and two successive values of the integral thus obtained are compared against a small tolerance, it is possible to reach the convergence criterion. The value of the integral thus calculated will have an inherent error besides a roundoff error, which will be proportional to the number of functional evaluations, but, for all practical purposes it may be acceptable. This procedure is tried with the Simpson's rule (20,000 panels) and with the Romberg method (tolerance = 10^{-4}), and the results are tabulated below:

a	Simpson's integral (relative error)	Romberg integral (relative error)
0.001	-0.8145640 (0.960890%)	-0.8145650 (0.960768%)
0.0001	-0.8214469 (0.124029%)	-0.8214516 (0.123458%)
0.00001	-0.8223462 (0.014688%)	-0.8223638 (0.012548%)
0.000001	-0.8224795 (-0.0015198%)	-0.8224844 (-0.0021156%)

Integrands having trigonometric functions sometimes may be tricky to integrate by numerical methods. An integrand $f(x) = \sin(x)$ to be integrated between 0 and 4π , if done by the Trapezoidal rule with 1, 2, or 4 panels produces an integral equal to zero (due to roundoff error, it is not exactly zero but a very small number in the order of 10^{-13}) where, in fact, the answer should be 6.283185. This happens because every integer multiple of π is a

root of the integrand and the Trapezoidal rule with 1, 2, or 4 panels uses evaluation of the integral at these points.

Development of a computer program

A subroutine **WEIGHT** is developed for supplying the values of zeros and weights of Legendre polynomials when called by the main program. It is listed in Appendix A.3.3.1. The subroutine **RULES**, listed in Appendix A.3.3.3, is developed in such a way that it is capable of calculating the integral of a given function using any of the following methods chosen by the user:

1. Trapezoidal method with and without end correction,
2. Simpson's rule with and without end correction,
3. Gauss-Legendre Quadrature method with up to 24 panels,
4. Romberg integration method with a maximum of 15 times interval halving, i.e., a maximum of $2^{15} = 32,768$ applications of the Trapezoidal rule.

The main program, **MAIN.FOR** listed in Appendix A.3.3.4, when executed, asks the user to select an option among four shown in the main menu, namely, the Trapezoidal rule, Simpson's method, Gauss Quadrature, and the Romberg method. If the Trapezoidal or Simpson's method is chosen, a secondary menu appears, and the user can select either of the methods with or without end corrections. After this is done, the user should type in input data, i.e., number of panels and the lower and upper limits of integration. In the case of the Romberg method, the user is asked to type in lower and upper limits and the tolerance.

An appropriate message is provided if the user unknowingly violates any of the following points in typing input data:

1. The lower limit is always less than the upper limit of integration.
2. Simpson's rule should only be used with an even number of panels.
3. Minimum number of panels can be one.
4. Gauss Quadrature can only be used with from 2 to 10, 12, 15, 16, 20, and 24 base points.
5. When the Romberg method does not converge, it may be due to an extremely small tolerance given.

Results include the number of panels/applications, the lower limit, the upper limit, and the value of the integral. On the top of every output, the name of the method chosen is shown. In the case of the Trapezoidal or Simpson's method, if end correction is chosen, it also appears along with the name of the method. All the results are shown on the screen, as well as stored

in a file called **DATA.OUT**, which is automatically created when the compiled program is executed.

The following Table 3.3.2 shows the results of integration of the typical integrand $f(x) = 1/x$ between the limit 1.0 and 150.5 by various methods using the program **MAIN.FOR**. The analytical solution is $\ln(x)$ and the exact answer is 5.013963084.

Table 3.3.2. Comparison of results of integration by various methods.

No. panels/ base points	Trapezoidal Rule				Simpson's Method				Gauss Quadrature		Romberg Method	
	with EC	error, %	without	error, %	with EC	error, %	without	error, %	Integral	error, %	Integral	error, %
10/10*	9.088714	-81.27	10.255690	-104.54	8.866733	-76.84	7.978809	-59.13	4.841808	3.43	5.013965	-3.8E-5
100/12	5.101108	-1.74	5.175750	-3.23	5.091455	-1.55	5.052844	-0.78	4.921971	1.84		
1000/15	5.014198	-4.6E-3	5.015818	-0.04	5.014157	-3.9E-3	5.013980	-3.4E-4	4.978422	0.71		
10000/20	5.013968	-9.8E-5	5.013986	-4.5E-4	5.013964	-1.8E-5	5.013964	-1.8E-5	5.006818	0.14		
20000/24	5.013968	-9.8E-5	5.013973	-2.0E-4	5.013967	-7.8E-5	5.013966	-5.8E-5	5.012002	0.04		

* Number of base points for Gauss Quadrature only.

N.B. Tolerance for Romberg method was taken as 10^{-6} , and the number of applications for convergence was 2048. For a tolerance equal to 10^{-4} , the number of applications was 1024.

It is observed from Table 3.3.2 that end correction was not very effective for this integrand. Though it improved the integral in the Trapezoidal rule to a small extent, it worsened the value in the Simpson's method. The power of the Romberg method is quite evident. With only 2048 applications of the Trapezoidal rule, it took just 11 extrapolations to reach an accuracy that was attained by the Simpson's method with 10,000 to 20,000 panels and was not attained at all by the Trapezoidal rule alone even with 20,000 panels. The Gauss Quadrature method yielded results better than those of the Trapezoidal and Simpson's methods with a comparable number of base points/panels.

So far, all the methods discussed have been shown to work with continuous functions as integrands. For the functional values available at discrete points, it may be necessary to smooth data by an appropriate least squares approximation and integrate the resulting regressed function.

References

1. **Abramowitz, M. and Stegun, I. A.**, *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*. Dover Publications, New York, 1972.
2. **Burden, R. L. and Faires, J. D.**, *Numerical Analysis*. Fifth edition, Prindle, Weber & Schmidt, Boston, 1993.
3. **Carnahan, B., Luther, H. A., and Wilkes, J. O.**, *Applied Numerical Methods*. John Wiley & Sons, New York, 1969.

4. **Dorn, W. S. and McCracken, D. D.**, *Numerical Methods with FORTRAN IV Case Studies*. John Wiley & Sons, New York, 1972.
5. **Hamming, R. W.**, *Numerical Methods for Scientists and Engineers*. Second Edition, International Student Edition, McGraw-Hill, Inc., Tokyo, Japan, 1973.
6. **Hildebrand, F. B.**, *Introduction to Numerical Analysis*. Second Edition, Dover Publications, New York, 1987.
7. **Hornbeck, R. W.**, *Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ, 310 pp., 1975.
8. **Kreyszig, E.**, *Advanced Engineering Mathematics*. Seventh edition, John Wiley & Sons, New York, 1993.
9. **Stroud, A. H. and Secrest, D.**, *Gaussian Quadrature Formulas*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

CHAPTER 3.4

Solution of Linear Algebraic Equations

Solution of a single algebraic equation, be it linear or not, yields its root(s) as we have seen in Chapter 3.2. In this chapter different methods will be discussed to solve a set of linear algebraic equations. An equation is called linear if each of its terms contains only one variable in its first power. Algebraic equations differ from transcendental equations in the fact that in the latter the independent variables appear in trigonometric, logarithmic, exponential, or some other form whereas in the former they appear alone.

In this connection, it is important to have a basic knowledge of matrix techniques because, in the solution of a set of equations, the matrix concept is used extensively. First, we will see how a set of equations can be represented by a rectangular array called a matrix. Notations used below will be adhered to throughout the chapter.

Let us consider the following set of linear algebraic equations:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = r_1 \quad 3.4.1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = r_2 \quad 3.4.2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = r_3 \quad 3.4.3$$

$$a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = r_4 \quad 3.4.4$$

In the above equations, the known coefficients are a_{11} , a_{12} , a_{13} , a_{14} , a_{21} , a_{22} , a_{23} , a_{24} , a_{31} , a_{32} , a_{33} , a_{34} , a_{41} , a_{42} , a_{43} , and a_{44} , and the known constant quantities on their right hand sides are r_1 , r_2 , r_3 , and r_4 . There are four

unknowns, which are x_1 , x_2 , x_3 , and x_4 . It is common knowledge that, in order to solve for four unknowns, there should be at least four independent equations containing these unknown variables. In the above example these equations are (3.4.1) through (3.4.4). The word **independent** is very important. Two equations are independent of each other if (i) one cannot be derived from the other by addition, subtraction, multiplication, or division, and (ii) they do not have the same root. When an equation is multiplied by a constant or by a variable or is subtracted from or added to an independent equation, it does not give rise to another independent equation.

A matrix is an array of numbers that may be of one row (called a **row vector**), of one column (called a **column vector**) or of a set of rows and columns (called a **rectangular** or a **square matrix**). Its concept arose from the necessity for bookkeeping in handling a large set of equations. Thus, in the case of solutions of a set of equations, the matrix always appears in square form where the number of rows and columns is equal, meaning that the number of independent equations is equal to that of unknowns. The above three equations [(3.4.1) through (3.4.4)] can be written in matrix form as follows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} \quad 3.4.5$$

This will be clear after the arithmetic operations with matrices are discussed. At this point it is convenient to learn that the left hand side of Eq. (3.4.5) containing the coefficients is a **square matrix** with four rows and four columns called the **coefficient matrix**. It is multiplied by a column vector of the unknowns (x_1 , x_2 , x_3 , and x_4) called the **solution vector**. The right hand side column vector containing r_1 , r_2 , r_3 , and r_4 is called the **constant vector**. The quantities inside a matrix are called its **elements**. Thus a_{11} , a_{12} ... a_{43} , a_{44} are the elements of the square matrix $[A]$. The matrix or the vector itself is denoted by an upper case letter whereas its elements by the lower case of the same letter. In this chapter, elements of the coefficient matrix $[A]$ are expressed as a_{ij} , i denoting the row positions and j the column positions.

The subscripts of the matrix elements bear specific meaning. The first subscript specifies the row position whereas the second identifies the position of the column of a particular element. Thus a_{23} is the element of matrix $[A]$ that is located at the second row and the third column. When there is only one subscript, the element either belongs to a column or a row vector.

Already the term "vector" has been used. A vector is a one-dimensional matrix. When the column dimension is unity, it is called a **column vector**. Similarly when the row dimension is unity, it is called a **row vector**.

The elements a_{11} , a_{22} , a_{33} , and a_{44} of matrix $[A]$ are called the **diagonal** elements, and this is often expressed as a_{ii} or a_{jj} . A matrix is called symmetric if the elements with row and column positions interchanged are equal, that is, $a_{ij} = a_{ji}$.

A **triangular** matrix is one in which all the elements either above or below the diagonal are zero. If the elements above the diagonal are zero, it is called the **lower triangular** matrix, and if all the elements below the diagonal are zero, it is called an **upper triangular** matrix. A **diagonal** matrix is one in which all the elements below and above the diagonal elements are zero. If the diagonal elements of a diagonal matrix are unity, it is termed **unit** or **identity** matrix.

A **banded** matrix has all the elements above and below a band on either side of the diagonal as zero. If the bandwidth is three, it is called a **tridiagonal** matrix, which is frequently encountered in the solution of partial differential equations as we will see in the next chapter. A tridiagonal matrix looks as follows:

$$\begin{bmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34} \\ & & a_{43} & a_{44} \end{bmatrix}$$

Arithmetic manipulations with matrices

Matrix operations such as addition, subtraction, multiplication and inversion are not exactly as simple as they are in plain arithmetic. In order to use matrix representation of a set of equations properly, it is necessary to know the rules for doing arithmetic manipulations with matrices. It will be easy to follow the rules if they can be explained at the element level. As before, the matrix as a whole is denoted by an upper case letter whereas its elements are denoted by a lower case one.

Addition of matrices

If $[A]$ and $[B]$ are any two matrices of identical row dimension and identical column dimension, their sum denoted by $[S]$ can be expressed as

$$[S] = [A] + [B]$$

or simply,

$$\begin{aligned} S &= A + B \\ &= B + A \end{aligned}$$

At element level,

$$\begin{aligned} s_{ij} &= a_{ij} + b_{ij} \\ &= b_{ij} + a_{ij} \end{aligned}$$

Subtraction between matrices

It is just like addition, but with a negative sign in one. Thus when [B] is subtracted from [A], if the subtracted result is denoted by [D],

$$\begin{aligned} [D] &= [A] - [B] \\ &= -[B] + [A] \end{aligned}$$

At element level,

$$\begin{aligned} d_{ij} &= a_{ij} - b_{ij} \\ &= -b_{ij} + a_{ij} \end{aligned}$$

Multiplication of matrices

If the product of [A] and [B] is denoted by [M], that is,

$$[M] = [A][B]$$

the column dimension of [A] should be equal to the row dimension of [B]. Thus if only [A] and [B] are two square matrices of identical row and column dimensions, their places can be interchanged to get [M], that is,

$$[M] = [B][A]$$

In general, when multiplication is done between nonsquare matrices, the resultant matrix [M] gets the row dimension of [A] and the column dimension of [B]. At the element level,

$$m_{ij} = \sum_{k=1}^n a_{ik} b_{kj} \quad 3.4.6$$

where $\mathbf{n}$ is the column dimension of $[A]$ and the row dimension of $[B]$. An example will be helpful in understanding this operation. Let us define $[A]$ and $[B]$ as

$$[A] = \begin{bmatrix} 3 & 2 & 1 & -1 \\ 1 & -2 & 2 & 1 \\ 4 & 1 & 1 & 2 \end{bmatrix} \quad [B] = \begin{bmatrix} 1 & 2 & 3 & 2 \\ -1 & 1 & 2 & 1 \\ 4 & 2 & 1 & 1 \\ 2 & 3 & 2 & 1 \end{bmatrix}$$

Here $[A]$ is a (3×4) matrix and $[B]$ a 4×4 matrix. Thus the column dimension of $[A]$ is equal to the row dimension of $[B]$, which is the necessary condition for multiplication. Thus $[M]$, the product of $[A]$ and $[B]$ should be a (3×4) matrix. The value of $\mathbf{n}$ in this case is 4.

Now using the expression (3.4.6), elements of $[M]$ can be calculated as

$$\begin{aligned} m_{11} &= a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} \\ &= (3)(1) + (2)(-1) + (1)(4) + (-1)(2) \\ &= 3 \end{aligned}$$

$$\begin{aligned} m_{12} &= a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} \\ &= (3)(2) + (2)(1) + (1)(2) + (-1)(3) \\ &= 7 \end{aligned}$$

$$\begin{aligned} m_{13} &= a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} + a_{14}b_{43} \\ &= (3)(3) + (2)(2) + (1)(1) + (-1)(2) \\ &= 12 \end{aligned}$$

$$\begin{aligned} m_{14} &= a_{11}b_{14} + a_{12}b_{24} + a_{13}b_{34} + a_{14}b_{44} \\ &= (3)(2) + (2)(1) + (1)(1) + (-1)(1) \\ &= 8 \end{aligned}$$

$$\begin{aligned} m_{21} &= a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} \\ &= (1)(1) + (-2)(-1) + (2)(4) + (1)(2) \\ &= 13 \end{aligned}$$

Similarly,

$$\begin{aligned} m_{22} &= 7; \quad m_{23} = 3; \quad m_{24} = 3 \\ m_{31} &= 11; \quad m_{32} = 17; \quad m_{33} = 19; \quad m_{34} = 12 \end{aligned}$$

Thus $[M]$ becomes

$$[M] = \begin{bmatrix} 3 & 7 & 12 & 8 \\ 13 & 7 & 3 & 3 \\ 11 & 17 & 19 & 12 \end{bmatrix}$$

when the identity or unit matrix [I] is multiplied with a square matrix [A], the product returns the square matrix itself. Thus

$$[A][I] = [A]$$

also

$$[I][A] = [A]$$

Inverse of a matrix

The arithmetic operation division does not exist in direct form for matrices, however, for a square matrix, its inverse does exist and is used for solution of a set of equations. Thus the inverse of square matrix [A] is denoted as $[A]^{-1}$ or simply A^{-1} , that is,

$$[A]^{-1} = \frac{1}{[A]}$$

When an inverse of a square matrix is multiplied by the original square matrix, the product becomes an identity matrix,

$$[A][A]^{-1} = [I]$$

also

$$[A]^{-1}[A] = [I]$$

If the inverse of a square matrix is again inverted, the original square matrix is obtained. It is often used as a check on the results of matrix inversion.

$$[[A]^{-1}]^{-1} = [A]$$

Transpose of a matrix

The transpose of a matrix [A], denoted as $[A]^T$ is obtained by interchanging rows and columns, that is, replacing a_{ij} by a_{ji} . The matrix [A] in the above example problem will have its following transpose, $[A]^T$

$$[A]^T = \begin{bmatrix} 3 & 1 & 4 \\ 2 & -2 & 1 \\ 1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix}$$

The transpose of sum of two matrices having identical row and column dimensions is the sum of transpose of the individual matrices, or, in other words, if $[A]$ and $[B]$ are two matrices with $(m \times n)$ dimensions,

$$[[A] + [B]]^T = [A]^T + [B]^T$$

Orthogonality of a matrix

A square matrix is said to be orthogonal if its transpose is identical to its inverse, that is,

$$[A]^T = [A]^{-1}$$

Determinant of a matrix

The determinant exists for a square matrix only. It is a value calculated by dividing the matrix into (2×2) **minors** of each of the elements of the first column and summing up the products formed by multiplying crosswise taking one element from each row starting from the top and one element from each column. The sign of each product is decided by the factor $(-1)^{i+k}$ where i and k are the i -th row and k -th column of the element considered.

For a (2×2) matrix $[A]$, its determinant (denoted as **det A** or **D**) is calculated as

$$D = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12}$$

For a (3×3) matrix, its determinant **D** shown below is calculated as follows:

$$D = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

If the row and column of the first element a_{11} are deleted the minor of a_{11} denoted by M_{11} , is given by

$$M_{11} = \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} = a_{22}a_{33} - a_{32}a_{23}$$

Again, deleting the row and column for the second element in the first column, the minor of a_{21} , that is M_{21} becomes

$$M_{21} = \begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} = a_{12}a_{33} - a_{32}a_{13}$$

Lastly, deleting the row and column for the third element in the first column, the minor M_{31} of a_{31} is

$$M_{31} = \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} = a_{12}a_{23} - a_{22}a_{13}$$

Now the determinant is calculated by the following expression:

$$\begin{aligned} D &= a_{11}M_{11} - a_{21}M_{21} + a_{31}M_{31} \\ &= a_{11}(a_{22}a_{33} - a_{32}a_{23}) - a_{21}(a_{12}a_{33} - a_{32}a_{13}) + a_{31}(a_{12}a_{23} - a_{22}a_{13}) \\ &= a_{11}a_{22}a_{33} - a_{11}a_{32}a_{23} + a_{21}a_{32}a_{13} - a_{21}a_{12}a_{33} + a_{31}a_{12}a_{23} - a_{31}a_{22}a_{13} \end{aligned}$$

In order to develop a general expression for calculating the determinant of $[A]$ having $(n \times n)$ dimensions, the **cofactor** of the element a_{ik} when the i -th row and the k -th column are deleted from the determinant $\mathbf{D}$, denoted as C_{ik} is defined as follows:

$$C_{ik} = (-1)^{i+k} M_{ik}$$

where M_{ik} is the minor of a_{ik} as defined earlier.

Thus the value of the determinant, $\mathbf{D}$ is given by

$$D = a_{1k}C_{1k} + a_{2k}C_{2k} + a_{3k}C_{3k} + \dots + a_{nk}C_{nk} \quad 3.4.7$$

where $\mathbf{k}$ may be any column number between 1 to n . In compact form, Eq. (3.4.7) is expressed as

$$D = \sum_{i=1}^n (-1)^{i+k} a_{ik} M_{ik} ; [k = \text{any number between 1 \& } n] \quad 3.4.8$$

The above expression (3.4.8) was developed considering any column. Similar expressions can be developed considering any row, and $\mathbf{D}$ is expressed by

$$D = \sum_{k=1}^n (-1)^{i+k} a_{ik} M_{ik} ; [i = \text{any number between 1 \& } n] \quad 3.4.9$$

Both the expressions given by (3.4.8) and (3.4.9) yield the same value for the determinant **D**.

Rank of a matrix

To check the existence of the inverse of a square matrix, the concept of **rank** of a matrix is essential. For an $(n \times n)$ matrix $[A]$, the inverse $[A]^{-1}$ exists if and only if the **rank** of $[A]$ is equal to **n**. It also tells about the existence of singularity. The matrix $[A]$ is singular if its rank is less than **n**. Thus the **rank** of a square matrix is defined as the maximum number of linearly independent row vectors of the matrix. The significance of the word **independent** has been explained in the beginning of the discussion about matrix.

After the above discussion on matrix and the manipulations thereof we are in a position to get down to the solution of a set of linear algebraic equations.

Solution of a set of linear algebraic equations

Before attempting to solve a set of linear algebraic equations with the matrix concept, let us solve just two such equations first and then three simultaneous equations manually. This will give an insight of arriving at the logical sequence of solving such equations employing different established methods discussed later in this chapter. In the first example, the two equations taken are as follows:

$$3x_1 + 2x_2 = 5$$

and

$$2x_1 - 5x_2 = -3$$

The usual practice is to eliminate the first variable by dividing and multiplying the first equation by some constants and subtracting or adding the transformed equation from or to the second equation. Once this operation is performed, the other variable can be directly computed. The third step is to calculate the variable that was eliminated in the first step by substituting the value of the second variable in any of the two original equations.

Thus, the first equation is divided by 3 and then multiplied by 2 so that the variable x_1 has the same coefficient in both the equations. The transformed equation becomes

$$2x_1 + 1.3333333x_2 = 3.3333333$$

When this equation is subtracted from the second equation, x_1 is eliminated to give

$$-6.3333333x_2 = -6.3333333$$

or

$$x_2 = 1.0000000$$

If this value is substituted in the first equation, x_1 is obtained as 1.0.

In the second example, let us solve three equations that are given as

$$4x_1 + 2x_2 + 3x_3 = 9$$

$$2x_1 - x_2 + x_3 = 2$$

$$3x_1 + 4x_2 - 2x_3 = 5$$

To eliminate x_1 , the first equation is divided by 4 and multiplied by 2, to get

$$2x_1 + x_2 + 1.5x_3 = 4.5$$

Now, the above transformed equation is subtracted from the second equation to yield

$$-2x_2 - 0.5x_3 = -2.5$$

To get another equation with only x_2 and x_3 , similar operations are performed on the first equation and the third equation, that is, the first equation is divided by 4 and then multiplied by 3, to obtain

$$3x_1 + 1.5x_2 + 2.25x_3 = 6.75$$

This is subtracted from the third equation to give

$$2.5x_2 - 4.25x_3 = -1.75$$

The above two transformed equations in x_2 and x_3 are again solved exactly the same way as the two equations were solved in the first example to give first the value of x_3 as 1.0 and then x_2 as 1.0. Substituting these values of x_2 and x_3 in any one of the three original equations, we get the value of x_1 as 1.0.

The advantage of this augmented matrix is that we now have only one rectangular matrix instead of a square matrix and a column vector. The Gauss method is used to reduce the augmented matrix $[A]^*$ to an upper triangular matrix by sequential logical operations. These logical operations are (1) multiplication or division of any equation by a constant and (2) adding or subtracting an equation from the other. It should be noted that these operations do not affect the solution vector. The following steps are followed to form the upper triangular matrix:

- Step 1. Make the diagonal element of the first row/equation as the pivot element and divide this row by it to normalize it,
- Step 2. Multiply the normalized row by the first element of the second row and subtract it from the second row,
- Step 3. Multiply the normalized row by the first element of the third row and subtract it from the third row,
-
- Step n. Multiply the normalized row by the first element of the n-th row and subtract it from the n-th row.

Steps 1 through n clear the entire first column to zero except for the first element of the first row, that is, a_{11} , which is left unchanged. In fact, the first row elements do not suffer any transformation throughout the process.

These steps (n-1) are repeated with the second row as the pivot row and its diagonal element as the pivot element to make the elements of the second column below the diagonal zero. Thus repeating these steps with the rest of the rows, all the elements below the diagonal become zero. Each time, the number of steps is reduced by one. The last augmented element divided by a'_{nn} after all these operations becomes equal to x_n . The rest of the unknowns can be calculated easily by back substitution. The notation $\mathbf{a}'$ differs from $\mathbf{a}$ in the fact that the former has suffered a transformation due to arithmetical operations on the particular row without affecting the solution vector. This will be clear from the following upper triangular matrix formed after the above operations:

$$\left[\begin{array}{cccccc} a_{11} & a_{12} & a_{13} & a_{14} & \dots & a_{1n} & a_{1,n+1} \\ & a'_{22} & a'_{23} & a'_{24} & \dots & a'_{2n} & a'_{2,n+1} \\ & & a'_{33} & a'_{34} & \dots & a'_{3n} & a'_{3,n+1} \\ & & & a'_{44} & \dots & a'_{4n} & a'_{4,n+1} \\ & & & & \dots & & \\ & & & & & a'_{nn} & a'_{n,n+1} \end{array} \right]$$

Back substitution

Starting with the last row, the value of x_n is obtained directly as

$$x_n = a'_{n,n+1} / a'_{nn}$$

From the one before the last row [(n-1)-th row]

$$\begin{aligned} x_{n-1} &= (a'_{n-1,n+1} - a'_{n-1,n} \cdot a'_{n,n+1}) / a'_{n-1,n-1} \\ &= (a'_{n-1,n+1} - a'_{n-1,n} \cdot x_n) / a'_{n-1,n-1} \end{aligned}$$

Similarly, from the (n-2)-th row,

$$x_{n-1} = (a'_{n-2,n+1} - a'_{n-2,n-1} \cdot x_{n-1} - a'_{n-2,n} \cdot x_n) / a'_{n-2,n-2}$$

Now, for any k-th row, a generalized expression can be developed by induction as

$$x_k = \left(a'_{k,n+1} - \sum_{i=1}^{n-k} a'_{k,i+k} \cdot x_{i+k} \right) / a'_{kk} \quad 3.4.11$$

In the course of the elimination process, the number of arithmetic operations can be reduced by observing that the operation of multiplying a row by a constant and subtracting the result from another row does not need to be done with the entire row. When the k-th row is taken as the pivot row, it implies that x_k is going to be eliminated. Thus the first (k-1) columns can be ignored including the k-th column. For this reason the index **jj** in Figure 3.4.1 starts at (k+1).

In the flow diagram shown in Figure 3.4.1, the subscript **k** refers to the number of the equation being subtracted from the other equations. It also identifies the variable that is being eliminated from the last (n-k) rows. The subscript **jj** refers to the number of the equation from which a variable is being eliminated, and **j** to the column position.

Example 3.4.1. Let us solve the three equations given earlier by the Gauss elimination.

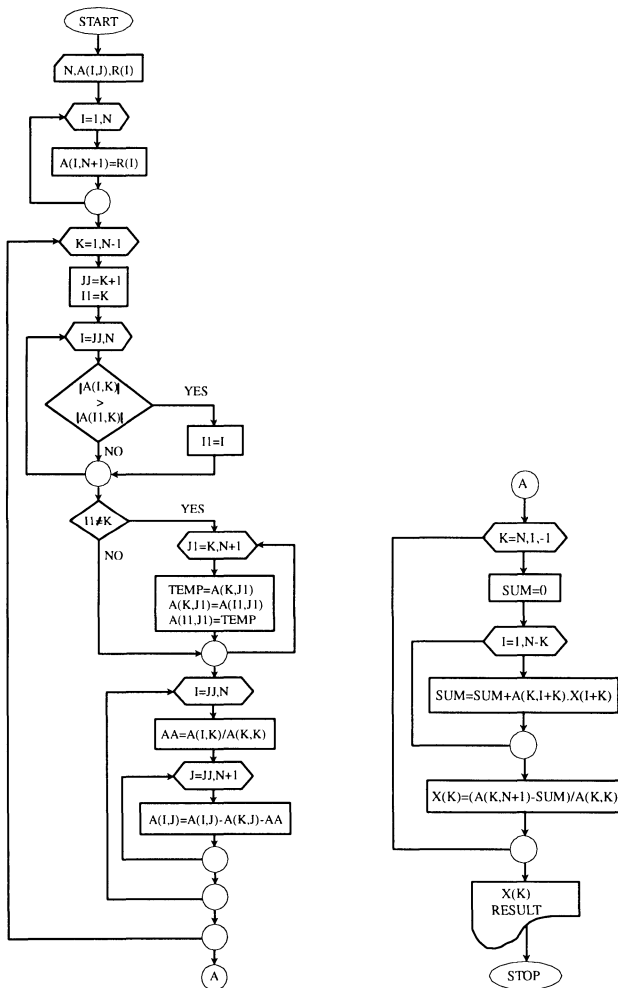
Solution. In matrix form the system can be expressed as

$$\begin{bmatrix} 4 & 2 & 3 \\ 2 & -1 & 1 \\ 3 & 4 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \\ 5 \end{bmatrix}$$

and the augmented matrix becomes

$$\begin{bmatrix} 4 & 2 & 3 & 9 \\ 2 & -1 & 1 & 2 \\ 3 & 4 & -2 & 5 \end{bmatrix}$$

Step 1. Divide the first row by its pivot element, that is, 4.0, to get the normalized row,



Formation of upper triangular matrix

Back substitution

Figure 3.4.1. Flow diagram for Gauss elimination method.

Step 2. Multiply the normalized row by the first element of the second row, and subtract it from the second row,

Step 3. Multiply the normalized row by the first element from the third row, and subtract it from the third row.

Steps (1) through (3) produce the following augmented matrix,

$$\begin{bmatrix} 4.0 & 2.0 & 3.0 & 9.0 \\ 0.0 & -2.0 & -0.5 & -2.5 \\ 0.0 & 2.5 & -4.25 & -1.75 \end{bmatrix}$$

Now, consider the second row as the pivot row and its diagonal element as the pivot element.

Step 4. To get the normalized row, divide the second row by -2,

Step 5. Multiply the normalized row by the second element of the third row, and subtract it from the third row; this gives

$$\begin{bmatrix} 4.0 & 2.0 & 3.0 & 9.0 \\ 0.0 & -2.0 & -0.5 & -2.5 \\ 0.0 & 0.0 & -4.875 & -4.875 \end{bmatrix}$$

From the last row, $-4.875x_3 = -4.875$, that is, $x_3 = 1.0$

Now, substituting this value of x_3 in the second row,

$$-2.0x_2 - 0.5x_3 = -2.5, \text{ that is, } x_2 = 1.0$$

These values of x_2 and x_3 , when substituting in the first row, give

$$4x_1 + 2x_2 + 3x_3 = 9, \text{ giving } x_1 = 1.0$$

Pivotal condensation

It is interesting to note that arithmetic operations in Gauss Elimination procedure and in other procedures as well, as we will gradually observe, an equation or a row of the coefficient matrix is always divided by its diagonal element, that is, by a_{kk} . Thus it is necessary to handle such situations to avoid a division by zero. One way of doing it is by interchanging rows. This does

not alter the solution vector, which can be observed from the matrix representation (3.4.10). Another way is to attribute a very small value to the diagonal element, which is otherwise zero. The first method is preferred to the second, at the cost of some more bookkeeping in programming, because the roundoff error sometimes produces meaningless results in the latter case especially when there is a large set of equations involved.

The **pivotal condensation** is a process of avoiding the pivot element to be very small, causing large roundoff error. This is also called maximization of pivot elements. It can be done by interchanging columns of the coefficient matrix to shift the largest element in its absolute value in the pivot row into the diagonal position. Thus the largest element in its absolute value becomes the pivot element. This operation is repeated with each new pivot row when necessary. An interchange in columns also implies an interchange in the position of the solution vector. The logic to programming such operations is very complex and has not been included in the programs developed for this chapter. On the other hand, the row interchange, that is, to rearrange the last $(n-k+1)$ rows so that the largest of the coefficients in its absolute value in the k -th column is in the k -th row, can be easily handled. This is called **partial pivotal condensation**, which has been implemented in the Gauss Elimination method. Such rearrangement is desirable even when the diagonal element is not zero.

In Figure (3.4.1) we can observe this partial pivotal condensation. After the loop starts with the counter k as 1, an auxiliary subscript $i1$ is introduced equal to k . First, a comparison is made between a_{ik} , which is just below the diagonal element a_{kk} , and a_{i1k} . If a_{ik} is larger in absolute value, $i1$ gets the value of i . The subscript $i1$ always represents the row number of the element in the k -th column, which is the largest one tested so far. The subscript i goes from $j1$ to n , that is, from $(k+1)$ to n . At the end of the loop, $i1$ identifies the largest element in the diagonal position after interchanging. This interchanging may not be necessary when the diagonal element is already the largest. In this case ($i1$ equal to k) such a possibility is tested, and the interchanging process is bypassed. It is now necessary to set i to its value before it is used as a subscript. This is done by continuing with the loop incrementing k by 1.

The interchanging process is done on pairs of values, one from row k and one from row $i1$. To interchange a_{kj1} and a_{i1j1} , a third temporary storage variable **TEMP** is introduced. It is done inside a loop using the subscript $j1$ which goes from k to $(n+1)$ as follows:

```
DO 10 J1=K,N+1
    TEMP = A(K,J1)
    A(K,J1) = A(I1,J1)
    A(I1,J1) = TEMP
10 CONTINUE
```

Gauss-Jordan elimination

In Gauss elimination method, an upper triangular matrix is formed and the values of the unknowns are calculated by back substitutions. The Gauss-Jordan elimination method is also a direct method for the solution of a set of simultaneous linear algebraic equations and also for the matrix inversion. The steps are identical to those used in Gauss elimination, the only difference here is to make all the elements on either side of the diagonal of the coefficient matrix zero and the diagonal elements unity, that is, to form an identity matrix having the same dimension as the coefficient matrix [A]. This is a step further ahead of the Gauss elimination method where the solution vector is obtained directly without any back substitution. Let us consider the following matrix representation of a set of linear algebraic equations:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \dots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots & a_{4n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ \dots \\ r_n \end{bmatrix} \quad 3.4.12$$

Now, if the coefficient matrix [A] is transformed into an identity matrix by arithmetic manipulations, the transformed column vector {R'} on the right hand side of Eq.(3.4.12) will represent the solution vector. This is precisely what is done in this method. Thus the steps pursued in Gauss elimination method will hold good, but, instead of the elimination process only down the pivot row, it should be done for all the rows except for the row under consideration, that is, the pivot row itself. For example, when the first row is the pivot, the elimination process should be applied to all the rest of the rows, just as in Gauss elimination method. Now, for the k-th row as pivot, the elimination process should start from the first row and terminate at the last or the n-th row, skipping the k-th row. This is achieved in the program by equating the pivot element A(K,K) equal to zero before the loop starts for the elimination process. The flow diagram is shown in Figure 3.4.2.

The partial pivotal condensation is not deliberately used in the program developed for this method and for the other methods as well. Anyone interested in adding this important feature can do it easily by analyzing the program developed for the Gauss elimination method. The less effective method of avoiding the division by zero when the diagonal element is zero has been used. With double precision, a small value of 10^{-10} for the zero diagonal element has been found effective for small number of equations. The efficiency of this inefficient method will be analyzed later in this chapter.

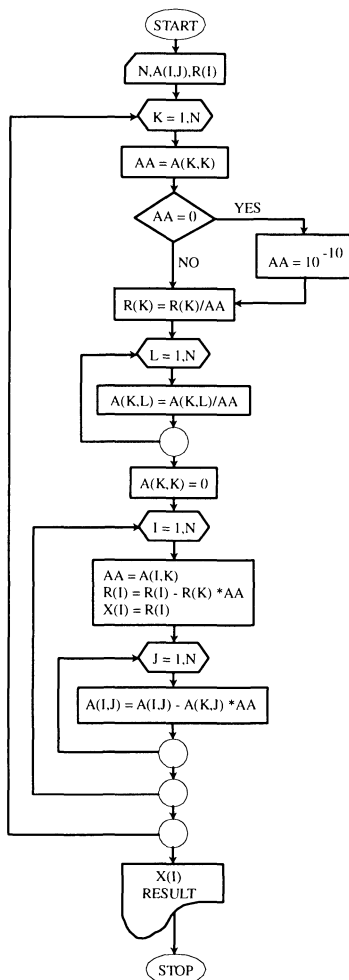


Figure 3.4.2. Flow diagram of the Gauss-Jordan elimination method.

Example 3.4.2. Let us solve the same problem given in Example 3.4.1. by the Gauss-Jordan elimination method.

Solution. It is not necessary to repeat all the steps taken in the Gauss elimination method to arrive at the upper triangular matrix. Rather we will start from the fifth step after dividing each row by its diagonal element. Just after the fifth step the matrix looks as below (here we have not constructed the augmented matrix as before without any logic, though it could be done as well):

$$\begin{bmatrix} 1.0 & 0.5 & 0.75 \\ 0.0 & 1.0 & 0.25 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2.25 \\ 1.25 \\ 1.00 \end{bmatrix}$$

Step 6. Now multiply the second row by the second element of the first row and subtract it from the first row,

Step 7. Make the third row as the pivot row and its diagonal element as the pivot element. Since this element is already 1.0, multiply the third row by the third element of the first row and subtract it from the first row, and

Step 8. Multiply the third row by the third element of the second row and subtract from the second row, finally to obtain

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

which directly gives the values of x_1 , x_2 , and x_3 as 1.0, 1.0, and 1.0, respectively.

Matrix inversion method

This is, in fact, the Gauss-Jordan elimination method where the inverse of the coefficient matrix is obtained as a byproduct. The inverse of a square matrix $[A]$ is defined by

$$[A][A]^{-1} = [I] \quad 3.4.13$$

The inverse $[A]^{-1}$ exists if $[A]$ is not singular, that is, if **det A** is not zero. Let us write the system of equations represented by (3.4.12) as,

$$[A]\{X\} = \{R\}$$

or,

$$\{X\} = [A]^{-1}\{R\} \quad 3.4.14$$

Thus the inverse of a matrix is useful in the solution of a set of simultaneous linear algebraic equations. The inverse of $[A]$ can be calculated using the definition in Eq. (3.4.13) as follows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \dots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots & a_{4n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \dots & a_{nn} \end{bmatrix} [A]^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

Now if the same elimination steps are used on $[A]$, as in the method of Gauss-Jordan elimination to turn it into an identity matrix, the left hand side of the above matrix representation will be $[I][A]^{-1}$, which is, in fact, the inverse of the matrix $[A]$, because, by definition, multiplication by an identity matrix returns the original matrix itself.

At the end of this transformation the transformed identity matrix $[A']$ on the right hand side will then be the inverted matrix $[A]^{-1}$. If extra bookkeeping is done to keep track of the column vector $\{R\}$, it will produce the solution vector $\{X\}$ at the end of the transformation along with producing the inverted matrix.

The flow diagram of this method will be identical to that of the Gauss-Jordan elimination method.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix} [A]^{-1} = \begin{bmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} & \dots & a'_{1n} \\ a'_{21} & a'_{22} & a'_{23} & a'_{24} & \dots & a'_{2n} \\ a'_{31} & a'_{32} & a'_{33} & a'_{34} & \dots & a'_{3n} \\ a'_{41} & a'_{42} & a'_{43} & a'_{44} & \dots & a'_{4n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a'_{n1} & a'_{n2} & a'_{n3} & a'_{n4} & \dots & a'_{nn} \end{bmatrix}$$

Example 3.4.3. Find the inverse of the coefficient matrix given in Example 3.4.1, and solve for the unknowns using the inverted matrix.

Solution. The system of equations is written according to Eq. (3.4.13) as follows:

$$\begin{bmatrix} 4 & 2 & 3 \\ 2 & -1 & 1 \\ 3 & 4 & -2 \end{bmatrix} [A]^{-1} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The steps will be exactly the same as the Gauss-Jordan elimination to convert $[A]$ to an identity matrix.

- Step 1. Divide the first row by its pivot element to make a normalized row,
 Step 2. Multiply the normalized row by the first element of the second row, and subtract it from the second,
 Step 3. Multiply the normalized row by the first element of the third row and subtract it from the third row. This will produce the following transformed system,

$$\begin{bmatrix} 1.0 & 0.5 & 0.75 \\ 0.0 & -2.0 & -0.5 \\ 0.0 & 2.5 & -4.25 \end{bmatrix} [A]^{-1} = \begin{bmatrix} 0.25 & 0.0 & 0.0 \\ -0.5 & 1.0 & 0.0 \\ -0.75 & 0.0 & 1.0 \end{bmatrix}$$

- Step 4. Let us now make the second row as the pivot row and its diagonal element as the pivot element. It is normalized by dividing it by the diagonal element, that is, by -2.0,
 Step 5. Multiply the normalized second row by the second element of the first row, and subtract it from the first row,
 Step 6. Multiply the normalized second row by the second element of the third row and subtract it from the third row. Thus we obtain,

$$\begin{bmatrix} 1.0 & 0.0 & 0.625 \\ 0.0 & 1.0 & 0.25 \\ 0.0 & 0.0 & -4.875 \end{bmatrix} [A]^{-1} = \begin{bmatrix} 0.125 & 0.25 & 0.0 \\ 0.25 & -0.5 & 0.0 \\ -1.375 & 1.25 & 1.0 \end{bmatrix}$$

- Step 7. Let us now make the third row as the pivot row and its diagonal element as the pivot element. It is normalized by dividing it by the diagonal element, that is, by -4.875,
 Step 8. Multiply the normalized third row by the third element of the first row, and subtract it from the first row,
 Step 9. Multiply the normalized third row by the third element of the second row, and subtract it from the second row. Thus we obtain,

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} [A]^{-1} = \begin{bmatrix} -0.05128 & 0.41026 & 0.12821 \\ 0.17949 & -0.43590 & 0.05128 \\ 0.28205 & -0.25641 & -0.20513 \end{bmatrix}$$

Thus the right hand side matrix of the above representation becomes the inverse of the coefficient matrix $[A]$. Now, from Eq. (3.4.14),

$$x_1 = (-0.05128)(9.0) + (0.41026)(2.0) + (0.12821)(5.0) = 1.00005$$

$$x_2 = (0.17949)(9.0) + (-0.43590)(2.0) + (0.05128)(5.0) = 1.00001$$

and

$$x_3 = (0.28205)(9.0) + (-0.25641)(2.0) + (-0.20513)(5.0) = 0.99998$$

These answers differ from the exact ones in the fifth decimal place due to truncation of the calculated quantities five places after the decimal.

All the methods described so far are the direct methods. These methods are finite in the sense that they need a predetermined (fixed) number of operations. For example, the Gauss elimination method requires $(4n^3 + 9n^2 - 7n)/6$ arithmetic operations, where n is the number of equations to be solved. When solving a large set of equations having the form of a banded matrix, these direct methods are most appropriate. In the next chapter we will see the development of such equations having banded coefficient matrices in the course of the solution of partial differential equation using the implicit method.

There are other techniques that are iterative, and the accuracy of such methods depends upon the number of iterations, which again depends upon the tolerance specified as the convergence criterion. The most widely used iterative method is called the Gauss-Seidel iteration.

Gauss-Seidel iteration

Let us explain this method using the first four equations (Eqs. 3.4.1 through 3.4.4) in this chapter, which were

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = r_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = r_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = r_3$$

$$a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = r_4$$

If the first equation is solved for x_1 , the second equation for x_2 , the third equation for x_3 , the fourth equation for x_4 , and so on, we obtain

$$x_1 = \frac{r_1 - a_{12}x_2 - a_{13}x_3 - a_{14}x_4}{a_{11}}$$

$$x_2 = \frac{r_2 - a_{21}x_1 - a_{23}x_3 - a_{24}x_4}{a_{22}}$$

$$x_3 = \frac{r_3 - a_{31}x_1 - a_{32}x_2 - a_{34}x_4}{a_{33}}$$

$$x_4 = \frac{r_4 - a_{41}x_1 - a_{42}x_2 - a_{43}x_3}{a_{44}}$$

By induction, the general expression can be written as

$$x_i = \frac{r_i - \sum_{\substack{j=1 \\ j \neq i}}^n a_{ij}x_j}{a_{ii}} \quad [i \text{ goes from } 1 \text{ to } n] \quad 3.4.15$$

In order to start the iterative scheme, the first guesses of $x_1, x_2, x_3, x_4, \dots, x_n$ are to be supplied. Usually these all are given a value of zero. In fact, this method is so powerful that the convergence is almost guaranteed irrespective of the value(s) of the initial guess(es) provided that the equations have proper characteristics. If the initial guesses are called $x_{1,0}, x_{2,0}, x_{3,0}, x_{4,0}, \dots, x_{n,0}$, the first iterated values are expressed as (using Eq. 3.4.15 and adding a second subscript for the number of iterations),

$$x_{1,1} = \frac{r_1 - a_{12}x_{2,0} - a_{13}x_{3,0} - a_{14}x_{4,0}}{a_{11}} \quad 3.4.16$$

$$x_{2,1} = \frac{r_2 - a_{21}x_{1,1} - a_{23}x_{3,0} - a_{24}x_{4,0}}{a_{22}} \quad 3.4.17$$

$$x_{3,1} = \frac{r_3 - a_{31}x_{1,1} - a_{32}x_{2,1} - a_{34}x_{4,0}}{a_{33}} \quad 3.4.18$$

$$x_{4,1} = \frac{r_4 - a_{41}x_{1,1} - a_{42}x_{2,1} - a_{43}x_{3,1}}{a_{44}} \quad 3.4.19$$

It is very interesting to note that the first iterated value, $x_{1,1}$, needs all the three guesses for its computation, whereas $x_{2,1}$ uses the first iterated value of x_1 , that is, $x_{1,1}$ and not the first guess $x_{1,0}$. Similarly $x_{3,1}$ is computed using the first iterated values of x_1 and x_2 ($x_{1,1}$ and $x_{2,1}$, respectively) instead of $x_{1,0}$ and $x_{2,0}$. Thus the iterative process consists of computing a new value of the unknown and replacing the old one with this most recent value. This is the special feature of Gauss-Seidel iteration that is responsible for its rapid convergence. For each iteration, it needs $2n^2$ arithmetic operations. Thus if the number of iterations for the Gauss-Seidel method is less than or equal to $n/3$, it is more efficient than the Gauss elimination method with respect to computer time used.

The iterative technique also suffers less roundoff error than the direct methods, which use the elimination process. In the elimination process the roundoff error is propagated throughout its $(4n^3 + 9n^2 - 7n)/6$ arithmetic operations. This feature sometimes becomes so important that, in spite of the total computation time exceeding that of Gauss elimination, the indirect method is resorted to. Also the iterative method is useful in solving a set of nonlinear equations. The flow diagram of this procedure is shown in Figure 3.4.3.

When all the unknowns have only one unique value, this iterative method does not seem to converge, unless the initial guesses are given as the values of the unknowns. If one attempts to solve the above example with initial values of x_1 , x_2 , and x_3 as 0.0, 0.0, and 0.0, even after 1000 iterations the scheme does not converge. If these initial guesses are given as 1.0, 1.0, and 1.0, it converges just after the first iteration. This is not always true though.

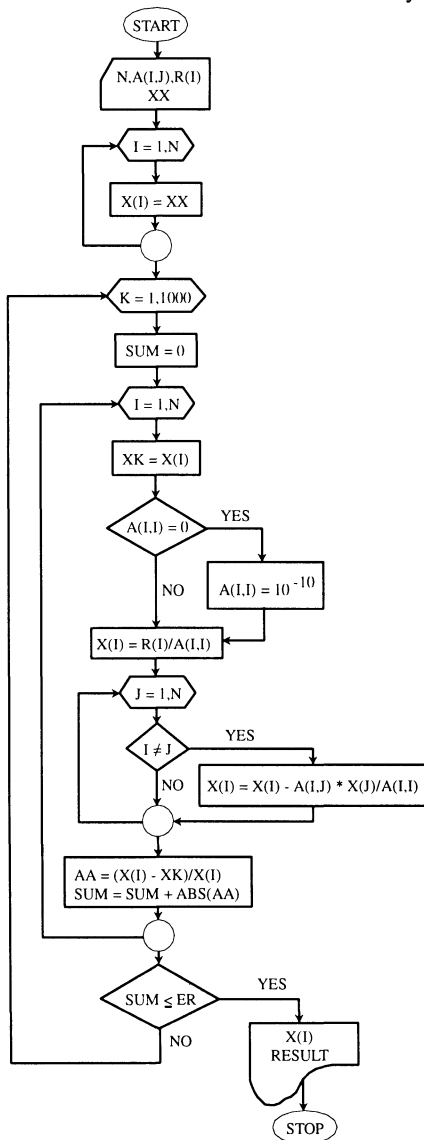


Figure 3.4.3. Flow diagram of the Gauss-Seidel iterative method.

The following set of equations, when solved by this method with an initial guess of 0.0 and a tolerance for convergence of 10^{-8} , converges after 11 iterations giving the solutions exact up to the eighth decimal place [exact solutions are $x_1 = 1.0$, $x_2 = 1.0$, and $x_3 = 1.0$]:

$$\begin{aligned}4x_1 - x_2 + x_3 &= 4 \\x_1 + 6x_2 + 2x_3 &= 9 \\-x_1 - 2x_2 + 5x_3 &= 2\end{aligned}$$

This and similar phenomena can be explained by pointing out that the diagonal terms must be dominant in order to facilitate convergence (its proof is omitted). In fact, the diagonal terms should be at least as large as the off-diagonal terms and actually larger in at least one case to guarantee a convergence. There are occasions where neither of these conditions is satisfied, and still the scheme converges.

Criteria for convergence

For terminating the iteration, two convergence criteria are generally used and these are (i) absolute and (ii) relative. According to the flow diagram shown in Figure 3.4.3, the subscript **k** is used for the number of iteration. Thus $x_{i,k}$ and $x_{i,k+1}$ represent the two successive iterated values of x_i . The absolute convergence criterion, then, can be expressed as the absolute difference between $x_{i,k+1}$ and $x_{i,k}$ less than or equal to the specified tolerance. On the other hand the relative convergence criterion, the absolute difference between $x_{i,k+1}$ and $x_{i,k}$, is divided by $x_{i,k+1}$.

The relative convergence criterion has been used in the program developed here. It is not practical to test the convergence criterion for each of the unknowns. Instead, the summation of the relative errors for all the unknowns can be tested just once against a suitable tolerance. This is precisely what has been demonstrated in the flow diagram. The variable **SUM** denotes this summation in the flow chart.

Relaxation in Gauss-Seidel iterative method

The process of convergence in Gauss-Seidel method can be improved, that is, it can be made faster by modifying the original algorithm. From the general equation (3.4.15), the current iterated values can be written as follows:

$$x_{i,k+1} = x_{i,k} + W(x_{i,k+1} - x_{i,k}) \quad 3.4.20$$

where w is unity.

Now, Eq.(3.4.20) can be written as

$$x_{i,k+1} = x_{i,k} + w \cdot \Delta x_i,$$

where for a four-equation system given by Eqs.(3.4.16) through (3.4.19),

$$\Delta x_1 = \frac{1}{a_{11}} \cdot (r_1 - a_{12}x_{2,k} - a_{13}x_{3,k} - a_{14}x_{4,k}) - x_{1,k}$$

$$\Delta x_2 = \frac{1}{a_{22}} \cdot (r_2 - a_{21}x_{1,k+1} - a_{23}x_{3,k} - a_{24}x_{4,k}) - x_{2,k}$$

$$\Delta x_3 = \frac{1}{a_{33}} \cdot (r_3 - a_{31}x_{1,k+1} - a_{32}x_{2,k+1} - a_{34}x_{4,k}) - x_{3,k}$$

$$\Delta x_4 = \frac{1}{a_{44}} \cdot (r_4 - a_{41}x_{1,k+1} - a_{42}x_{2,k+1} - a_{43}x_{3,k+1}) - x_{4,k}$$

The terms Δx_1 , Δx_2 , Δx_3 , and Δx_4 are the correction factors that when added to a value at the k -th iteration, return corrected/improved values at the next iteration.

The relaxation w is a real variable whose value varies from $0 < w < 1$ in case of underrelaxation. If w is unity we get back the original/unmodified Gauss-Seidel method. If it is greater than unity, it is the case of overrelaxation. The relaxation can only be negative when the Gauss-Seidel method is diverging in a nonoscillatory manner.

There is no way to calculate the optimum value of w . Only by the method of trial and error, can its optimum value be determined. The generalized modified Gauss-Seidel method is expressed by

$$x_{i,k+1} = x_{i,k} + w \cdot \Delta x_i, \quad 3.4.21$$

where

$$\Delta x_i = \bar{x}_i - x_{i,k}$$

and

$$\bar{x}_i = \frac{1}{a_{ii}} \left(r_i - a_{i1}x_{1,k+1} - \dots - a_{i,i-1}x_{i-1,k+1} - a_{i,i+1}x_{i+1,k} - \dots - a_{in}x_{n,k} \right)$$

where i goes from 1 to n . For $w = 1.0$, we go back to unmodified Gauss-Seidel method.

The modified Gauss-Seidel method should be used with caution. It has not been programmed here. The interested reader can modify the program listed in Appendix A.3.4.1 to incorporate the relaxation. The relaxation w

should be read by the main program so that it can be varied to analyze the effect of under- and/or overrelaxation on the speed of convergence.

Gauss elimination method in solving banded matrix

The definition of a banded or tridiagonal matrix was given earlier in this chapter. Unlike the solution of a set of equations having a square matrix, solution of a set having a banded matrix requires many fewer computations and also suffers much less roundoff error. As a result, without any pivotal condensation, several thousand such equations can be solved with utmost precision.

Instead of defining the coefficient matrix elements by two dimensions, it will be convenient to define them separately as the **d** diagonal, **a** above-diagonal and **b** below-diagonal row-wise 1-D elements, as shown below:

$$\begin{bmatrix} d_1 & a_1 & & & & & \\ b_2 & d_2 & a_2 & & & & \\ & b_3 & d_3 & a_3 & & & \\ & & b_4 & d_4 & a_4 & & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & & b_{n-1} & d_{n-1} & a_{n-1} \\ & & & & & b_n & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ \dots \\ r_{n-1} \\ r_n \end{bmatrix}$$

This single subscript notation of the elements in the coefficient matrix helps in facilitating the application of the Gauss elimination method, which considers the rows one at a time. If the Gauss elimination method is applied starting the first row, it is noted that, in each step, one below-diagonal element is eliminated. Thus once the bottom row is reached, the transformed banded matrix takes the form given next:

$$\begin{bmatrix} 1 & a'_1 & & & & & \\ & 1 & a'_2 & & & & \\ & & 1 & a'_3 & & & \\ & & & 1 & a'_4 & & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & & 1 & a'_{n-1} & \\ & & & & & 1 & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} r'_1 \\ r'_2 \\ r'_3 \\ r'_4 \\ \dots \\ r'_{n-1} \\ r'_n \end{bmatrix}$$

The value of x_n is directly equal to r'_n , and, by back substitution, the rest of the unknowns (x_{n-1} to x_1) can easily be computed. A flow diagram of this algorithm is given in Figure 3.4.4.

The number of basic arithmetic operations to solve a tridiagonal matrix is of the order n as compared to n^3 in the case of solving a full matrix. Here more emphasis is given on solving a tridiagonal matrix where there is only one element each below and above the diagonal elements because, in the course of solving partial differential equations numerically, such a type of matrix arises. Any set of equations in the form of a banded matrix having a wider bandwidth than discussed here can be solved by extending the technique learned so far.

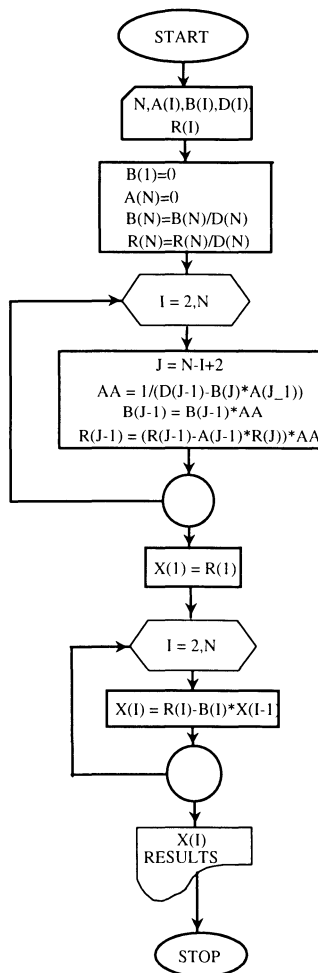


Figure 3.4.4. Gauss elimination algorithm for tridiagonal matrix.

Development of computer programs

The main program, **MAIN.FOR** developed in FORTRAN language to solve a set of simultaneous linear algebraic equations is listed in Appendix A.3.4.2. It calls a subroutine named **SOLVE** to solve the linear equations by (i) Gauss elimination, (ii) Gauss-Jordan elimination, (iii) Matrix inversion, and (iv) Gauss-Seidel methods. The subroutine **SOLVE.FOR** is listed in Appendix A.3.4.1. A double precision arithmetic is used for all the variables.

The data are to be written in free format to a file named **MATRIX.IN**, which is to be created before compiling the program. The data input should be made with (i) the number of equations (integer number), (ii) all the row-wise values of the coefficient matrix (real numbers), and (iii) all the values of the constant vectors (real numbers). A sample data input file has been listed in Appendix A.3.4.3.

In the subroutine **SOLVE**, the partial pivotal condensation has been applied only to the Gauss elimination method, but it keeps room for applying the same concept to the rest of the methods. It depends on the readers to incorporate this concept.

Another program has been developed to solve equations forming a tridiagonal coefficient matrix. It is listed in Appendix A.3.4.5 as **BANDED.FOR**. It calls a subroutine **TRIDGL** listed in Appendix A.3.4.4 as **TRIDGL.FOR**. Its flow diagram is shown in Figure 3.4.4.

The input data to this program should be provided by an input data file named **MATRIX.TRI**. Data should be entered in the following manner: (i) the number of equations (integer number), (ii) all the elements above the diagonal, (iii) all the elements below the diagonal, and (iv) all the diagonal elements.

There will be one value short in both the above- and below-diagonal elements [e.g., $A(N)$ and $B(1)$], but two real values should be attributed to them in the data file anyway. It does not matter what real values are given for them. The subroutine takes care of them by forcing them to zero. A sample data input file **MATRIX.TRI** is listed in Appendix A.3.4.6.

When the main program is compiled and executed, a main option menu appears on the screen. The user can choose any one of the four methods just discussed. In the case of the matrix inversion method, the results show the inverted matrix first and when <RETURN> is pressed, it shows the solution vectors. When the Gauss-Seidel method is chosen, the user is asked to supply the initial guess and the tolerance. If the scheme does not converge, the user is given the opportunity to change the initial guess and/or tolerance to try again without quitting the program. All the results are shown on the screen as well as stored automatically in a data file called **DATA.OUT**. Every time the program is executed, the previously created **DATA.OUT** is wiped out to give

Table 3.4.1. The effect of substitution of zero pivot element(s) by very small numbers (10^{-5} to 10^{-10}).

Unknowns	Exact values	Gauss-Jordan Method				Matrix Inversion Method			
		Number of zero diagonal elements				Number of zero diagonal elements			
		3		4		3		4	
		Computed value	Error, %	Computed value	Error, %	Computed value	Error, %	Computed value	Error, %
x_1	4.19925336	4.199253	0.0	4.199253	8.81E-6	4.199253	0.0	4.199253	0.0
x_2	0.49547354	0.495474	-8.07E-6	0.495474	-5.65E-5	0.495474	-8.07E-6	0.495474	-6.05E-6
x_3	-2.03082923	-2.030829	-4.92E-7	-2.030829	8.86E-6	-2.030829	-4.92E-7	-2.030829	0.0
x_4	7.75893900	7.758939	-2.58E-7	7.758939	-7.73E-7	7.758939	-2.58E-7	7.758939	-1.29E-7
x_5	8.47673810	8.476738	-1.18E-7	8.476738	-1.06E-6	8.476738	-1.18E-7	8.476738	-1.18E-7
x_6	2.74240310	2.742403	0.0	2.742403	8.75E-6	2.742403	0.0	2.742403	0.0
x_7	7.04432074	7.044321	-1.42E-7	7.044321	-1.42E-7	7.044321	-1.42E-7	7.044321	-1.42E-7
x_8	4.88247301	4.882473	-4.10E-7	4.882473	-1.64E-6	4.882473	-4.10E-7	4.882473	-2.05E-7

room for the contents of the most recent **DATA.OUT**. This file can be printed, copied, or just be looked at like any other DOS files.

It has been discussed earlier in this chapter that, in the event of any of the diagonal element being zero, a very small value of it (10^{-10}) can save the situation. The effect of doing this is illustrated in Table 3.4.1. In the Gauss elimination method, it does not alter the solution vector because of the partial pivotal condensation used. In the case of the other two methods, namely Gauss-Jordan and Matrix Inversion, where partial pivotal condensation has not been used, the relative error is shown. It should be remembered that, in accordance with the number of equations being solved, this error varies. In general, the greater the number of equations, the higher is the roundoff error. The Gauss-Seidel method does not seem to work by substituting any zero diagonal element with a small number. Thus the partial pivotal condensation may be the only answer for this method to deal with such zero pivot elements.

The results from the Table 3.4.1 clearly indicate that substitution of zero diagonal elements by small numbers on the order of 10^{-5} to 10^{-10} produces very small relative errors in the solution of a set of eight equations (Appendix A.3.4.3 lists the data for this set). There was no error up to the eighth decimal place when one or two diagonal elements were zero. This observation will certainly vary with the number, type, and arrangement of the equations being solved.

References

1. **Burden, R. L. and Faires, J. D.**, *Numerical Analysis*. Fifth edition, Prindle, Weber & Schmidt, Boston, 1993.
2. **Carnahan, B., Luther, H. A., and Wilkes, J. O.**, *Applied Numerical Methods*. John Wiley & Sons, New York, 1969.
3. **Dorn, W. S. and McCracken, D. D.**, *Numerical Methods with FORTRAN IV Case Studies*. John Wiley & Sons, New York, 1972.
4. **Hamming, R.W.**, *Numerical Methods for Scientists and Engineers*. Second Edition, International Student Edition, McGraw-Hill, Inc., Tokyo, Japan, 1973.
5. **Hildebrand, F. B.**, *Introduction to Numerical Analysis*. Second Edition, Dover Publications, New York, 1987.
6. **Hornbeck, R. W.**, *Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ, 310 pp., 1975.
7. **Kreyszig, E.**, *Advanced Engineering Mathematics*. Seventh edition, John Wiley & Sons, New York, 1993.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

CHAPTER 3.5

Least Squares Approximation

Even when the mathematical model of a physical process is developed using an analytical approach, the coefficients of such models are often estimated as functions of some experimental variables. Least squares approximation to a set of data points is a very handy tool for expressing such coefficients. We will start with fitting data to the multilinear function and show how the same concept can be used in the case of linear, polynomial, and other functions. In this discussion, it should be remembered that only those equations of the curves that can be easily linearized using known techniques will be considered.

In solving a problem that involves sets of variables bearing some inherent relationship among themselves, the statistical aspect of the problem then becomes one of arriving at the best estimate of the relationship among such variables. For example, in drying of an agricultural product the varying moisture content can be measured against time of its exposure to heated air at a certain temperature and relative humidity. It may be of interest to develop a method of estimation, i.e., a procedure for estimating the moisture content as a function of time from experimental information. This relationship, fitted to a set of experimental data, is characterized by a prediction equation called a **regression equation**, which may be linear or nonlinear with respect to the independent variable, which is time in our example. In another case, if the moisture content is desired to be a function of not only the drying time but also the condition of the heated air, that is, temperature and relative humidity, the regression equation becomes multilinear. The moisture content is the dependent variable, and time, temperature, and relative humidity are the independent ones.

The above drying problem can be solved in two ways. The first one is to fit the data on moisture content and time to a suitable linear or nonlinear regression and later fit the values of each of the coefficients of such an

equation and temperature and/or relative humidity to a multilinear or to a suitable linear or nonlinear regression. The second method is to fit the data on moisture, time, temperature and relative humidity to a multilinear regression.

Given a set of points, a straight line passing as close as possible to each point may very well be more useful than some complicated curve passing exactly through each point. It depends on the experimental data, which theoretically should fall along a straight line but due to errors in observation fail to do so. In the method of least squares the necessary measure of "as close as possible" is taken to be the criterion, and this criterion is always applied.

Though it is intended to use the multilinear regression as the principal format and derive the procedures for the various other regressions as special cases it, the explanation of the least square method will be more appropriate with the example of the linear regression.

Linear Regression

For linear regression the equation is

$$y = b_0 + b_1x \quad 3.5.1$$

where, b_0 and b_1 are the coefficients of Eq. (3.5.1), the values of which should be estimated by the least square method so that the error between the experimental and estimated values of y comes to a minimum. There is one independent variable x here. Thus it is observed that the number of coefficients is always one more than the number of independent variable(s).

If there are n experimental data points with n values of y , there should be n values of the independent variable x . Again considering Eq. (3.5.1), it can be rewritten in the following form :

$$y_i - b_0 - b_1x_i = e_i \neq 0$$

where y_i are the experimental values of the dependent variable, and e_i are the errors between the estimated and the experimental y values. These errors in general should not be zero owing to the fact that it is almost impossible to have an exact fit with experimental observations.

If this discrepancy, e_i , for each point of the set is computed, and the sum of the squares of these quantities is formed in order to prevent large positive and large negative e 's canceling each other and thereby giving an unwarranted impression of accuracy, we obtain

$$E = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - b_0 - b_1 x_i)^2 \quad 3.5.2$$

The quantity E is the measure of how well the calculated values fit the experimental data. The least-square criterion is that the coefficients b_0 and b_1 should be chosen so as to make the sum of squares of the deviations E as small as possible.

The usual conditions for minimizing a function are applied to do this, which are to equate the first partial derivatives of the left hand side variable of Eq. (3.5.2) with respect to each of the coefficients on its right hand side to zero. This would give two normal equations against two unknown coefficients. These normal equations are obtained as explained below :

$$\frac{dE}{db_0} = 2 \sum_{i=1}^n (y_i - b_0 - b_1 x_i)(-1) = 0 \quad 3.5.3$$

and

$$\frac{dE}{db_1} = 2 \sum_{i=1}^n (y_i - b_0 - b_1 x_i)(-x_i) = 0 \quad 3.5.4$$

Equation (3.5.3), when rearranged, yields

$$nb_0 + b_1 \sum_{i=1}^n x_i = \sum_{i=1}^n y_i$$

and Eq. (3.5.4) yields

$$b_0 \sum_{i=1}^n x_i + b_1 \sum_{i=1}^n x_i^2 = \sum_{i=1}^n x_i y_i$$

The above two normal equations can be expressed in matrix form as follows :

$$\begin{bmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \end{bmatrix} \quad 3.5.5$$

The FORTRAN source code of the main program for linear regression, which calls the multilinear regression subroutine **REGMLT** and the

subroutine of Gauss-Jordan elimination method **GAUSS** for solution of simultaneous linear algebraic equations given in Eqs. (3.5.5), is provided in Appendix A.3.5.5. This solves for the regression coefficients b_0 and b_1 .

Multilinear Regression

Let us suppose that we wish to fit a multilinear equation to our experimental data of the form :

$$y = b_0 + b_1x_1 + b_2x_2 + \dots + b_kx_k \quad 3.5.6$$

where $b_0, b_1, b_2, \dots, b_k$ are the coefficients of the multilinear equation (3.5.6), the values of which should be estimated by the least square method so that the error between the experimental and estimated values of y comes to a minimum. There are k independent variables $x_1, x_2, x_3, \dots, x_k$ here. So it is observed that the number of coefficients is always one more than the number of independent variables, i.e. $k+1$ here.

If there are n experimental data points, with n values of y there should be n values of each of the k independent variables. We denote these variables by $x_{11}, x_{12}, x_{13}, \dots, x_{1n}; x_{21}, x_{22}, x_{23}, \dots, x_{2n}; \dots x_{k1}, x_{k2}, x_{k3}, \dots, x_{kn}$. A linear regression will be a special case of the multilinear by forcing k to be equal to one and x_1 to be the only independent variable denoted by x . A polynomial regression of k -th order will be a special case of the multilinear regression by equating x_1 by x , x_2 by x^2 , x_3 by x^3 , and x_k by x^k . Again considering Eq. (3.5.6), it can be rewritten in the following form :

$$y_i - (b_0 + b_1x_{1i} + b_2x_{2i} + \dots + b_kx_{ki}) = e_i \neq 0$$

where y_i are the experimental values of the independent variable and e_i are the errors between the estimated and the experimental y values.

If this discrepancy e_i for each point of the set is computed and the sum of the squares of these quantities is formed, we obtain

$$E = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n \left(y_i - b_0 - \sum_{j=1}^k b_j x_{ji} \right)^2 \quad 3.5.7$$

The quantity E is the measure of how well the calculated values fit the experimental data. The least-square criterion is that the coefficients $b_0, b_1, \dots, b_k$ should be chosen so as to make the sum of squares of the deviations E as small as possible.

To minimize the function, the first partial derivatives of the left hand side variable of Eq. (3.5.7) with respect to each of the coefficients on its right hand side are equated to zero. This would give $(k+1)$ normal equations against $(k+1)$ unknown coefficients. In matrix form it will be a $(k+1 \times k+1)$

square matrix, which can be solved for the coefficients using a suitable subroutine. These normal equations will appear as given below :

$$\begin{aligned}
 nb_0 + b_1 \sum x_{1i} + b_2 \sum x_{2i} + \dots + b_k \sum x_{ki} &= \sum y_i \\
 b_0 \sum x_{1i} + b_1 \sum x_{1i}^2 + b_2 \sum x_{1i}x_{2i} + \dots + b_k \sum x_{1i}x_{ki} &= \sum x_{1i}y_i \\
 b_0 \sum x_{2i} + b_1 \sum x_{2i}x_{1i} + b_2 \sum x_{2i}^2 + \dots + b_k \sum x_{2i}x_{ki} &= \sum x_{2i}y_i \\
 &\dots\dots\dots \\
 b_0 \sum x_{ki} + b_1 \sum x_{ki}x_{1i} + b_2 \sum x_{ki}x_{2i} + \dots + b_k \sum x_{ki}^2 &= \sum x_{ki}y_i
 \end{aligned}$$

Now the above equations can be written in the matrix form as follows:

$$\begin{bmatrix}
 n & \sum x_{1i} & \sum x_{2i} & \dots & \sum x_{ki} \\
 \sum x_{1i} & \sum x_{1i}^2 & \sum x_{1i}x_{2i} & \dots & \sum x_{1i}x_{ki} \\
 \sum x_{2i} & \sum x_{2i}x_{1i} & \sum x_{2i}^2 & \dots & \sum x_{2i}x_{ki} \\
 \dots & \dots & \dots & \dots & \dots \\
 \sum x_{ki} & \sum x_{ki}x_{1i} & \sum x_{ki}x_{2i} & \dots & \sum x_{ki}^2
 \end{bmatrix}
 \begin{bmatrix}
 b_0 \\
 b_1 \\
 b_2 \\
 \dots \\
 b_k
 \end{bmatrix}
 =
 \begin{bmatrix}
 \sum y_i \\
 \sum x_{1i}y_i \\
 \sum x_{2i}y_i \\
 \dots \\
 \sum x_{ki}y_i
 \end{bmatrix}$$

3.5.8

If the above matrix representation is expressed as

$$[A]\{b\} = \{g\}$$

the solution for the regression coefficients can be written as

$$\{b\} = [A]^{-1}\{g\}$$

where $[A]^{-1}$ is the inverse of matrix $[A]$.

The FORTRAN source codes of subroutine **REGMLT** for this regression and another subroutine **GAUSS** for the solution of simultaneous linear algebraic equations using Gauss-Jordan elimination method are given in Appendices A.3.5.1. and A.3.5.3. The source code of the main program is listed in Appendix A.3.5.6.

Trigonometric Regression

The trigonometric function can have one or more terms in sine or cosine or tangent. These are generally useful when the input data vary according to one of them. For example, temperature variation during the day due to solar radiation may follow the sine function. Here the following trigonometric regression will be discussed — any other trigonometric regression will be just a matter of minor modifications on it:

$$y = b_0 + b_1 \sin(wx) + b_2 \sin(2wx) + \dots + b_k \sin(kwx) \quad 3.5.9$$

The matrix representation (3.5.8) will be used in the case of trigonometric regression. Only x_{1i} , x_{2i} , ..., x_{ki} should be replaced by $\sin(wx_i)$, $\sin(2wx_i)$, ..., $\sin(kwx_i)$, respectively.

The FORTRAN source code of the main calling program to do the above trigonometric regression is given in Appendix A.3.5.7.

Page's Equation

This equation first used by Page in 1946 is found to be very appropriate to fit the moisture data with time in a drying process. It is given by

$$y = e^{b_0 \cdot x^{b_1}} \quad 3.5.10$$

Equation (3.5.10) can be linearized by taking $\log_e$ two times on both its sides. Thus,

$$\ln(y) = b_0 \cdot x^{b_1}$$

$$\text{and} \quad \ln(\ln(y)) = \ln(b_0) + b_1 \ln(x) \quad 3.5.11$$

Now, Eq. (3.5.11) can be written as

$$Y = B_0 + B_1 X \quad 3.5.12$$

where

$$\begin{aligned} Y &= \ln(\ln(y)) \\ X &= \ln(x) \\ b_0 &= e^{B_0} \quad \text{and} \\ b_1 &= B_1 \end{aligned}$$

With the transformed variables X and Y Eq. (3.5.12) can be treated exactly like a linear function, and values of B_0 and B_1 can be estimated the same way described earlier. Here the restrictions would be that data on the dependent variable x and the independent variable y cannot have negative

values. Also y cannot be a fraction. Many other nonlinear functions, such as geometric ($y = b_0 x^{b_1}$), exponential ($y = b_0 e^{b_1 x}$), etc. can be linearized.

The FORTRAN source code for Page's regression has been listed in Appendix A.3.5.8.

Asymptotic Regression

The asymptotic function is given by the following equation:

$$y = b_0 + b_1 \cdot e^{b_2 \cdot x} \quad 3.5.13$$

This can be written as

$$y = b_0 + b_1 p^x \quad 3.5.14$$

where

$$p = e^{b_2} \text{ or } b_2 = \ln(p)$$

Linearization of Eq. (3.5.14) is not obvious as in the other cases just discussed.

If the function $f(p) = p^x$ is considered analytical around $p = r_0$, where r_0 is the first estimate of p , this function can be expanded up to the first two terms by the Taylor Series to obtain

$$\begin{aligned} f(p) &= f[r_0 + (p - r_0)] = p^x \\ &= f(r_0) + (p - r_0)f'(r_0) + \dots \\ &= r_0^x + (p - r_0) \frac{d}{dx}(r_0^x) \\ &= r_0^x + (p - r_0) \cdot x \cdot r_0^{x-1} \end{aligned} \quad 3.5.15$$

where, f' represents the first derivative of the function f .

Now, substituting the expression for p^x from Eq. (3.5.15) in Eq. (3.5.14), we get

$$\begin{aligned} y &= b_0 + b_1 r_0^x + b_1 (p - r_0) \cdot x \cdot r_0^{x-1} \\ &= b_0 + b_1 x_1 + b_{11} x_2 \end{aligned} \quad 3.5.16$$

where

$$\begin{aligned} x_1 &= r_0^x \\ x_2 &= x \cdot r_0^{x-1} \end{aligned}$$

and

$$b_{11} = b_1 (p - r_0) \quad 3.5.17$$

Initially, r_0 was the first approximation of the coefficient p , which should have a positive value. Now, to calculate the coefficient, b_{11} , p in Eq.(3.5.17) was replaced by another guess for p . If we call this second approximation r_1 , from Eq. (3.5.17) we get

$$r_1 = r_0 + \frac{b_{11}}{b_1} \quad 3.5.18$$

In order to get this scheme to work, the first guess of p , that is, r_0 should be provided with some accuracy. It has been found that if r_0 is calculated using the following expressions, the above scheme works satisfactorily. For odd number of observations (n is odd),

$$r_0 = \frac{-y(2) + y(3) - y(4) + \dots + y(n)}{-y(1) + y(2) - y(3) + \dots + y(n-1)}$$

and for even number of observations (n even),

$$r_0 = \frac{-y(2) + y(3) - y(4) + \dots + 2y(n-1) - y(n)}{-y(1) + y(2) - y(3) + \dots + 2y(n-2) - y(n-1)}$$

Now, Eq. (3.5.16) can be solved exactly as a multilinear equation with two independent variables, x_1 and x_2 . In the matrix representation (3.5.8), if k is made equal to 2, it will estimate the coefficients b_0 , b_1 , and b_{11} with the first guess of p equal to r_0 . After this first iteration, the absolute difference of the sum of error squares between the calculated and experimental values of y during two consecutive iterations is checked against a given small number (ERROR) in the order of 10^{-6} for convergence. Until this criterion that this difference is less than or equal to ERROR is met, the guess of p is corrected by Eq. (3.5.18) at each iteration. When the convergence criterion is satisfied, it is assumed that the calculated values of the regression coefficients are acceptable. Then the coefficient b_2 is obtained by $\ln(r_1)$ or $\ln(p)$.

Logistic Regression

The logistic equation is expressed as

$$\begin{aligned} y &= \frac{b_0}{1 + b_1 \cdot e^{b_2 \cdot x}} \\ &= \frac{b_0}{1 + b_1 \cdot p^x} \end{aligned} \quad 3.5.19$$

where

$$p = e^{b_2} \text{ or } b_2 = \ln(p)$$

Equation (3.5.19) can be expressed as

$$\frac{1}{y} = \frac{1}{b_0} + \frac{b_1}{b_0} \cdot p^x$$

or

$$Y = B_0 + B_1 \cdot p^x \quad 3.5.20$$

where

$$b_0 = \frac{1}{B_0}$$

$$b_1 = B_1 \cdot b_0 = \frac{B_1}{B_0}$$

Equation (3.5.20) can be solved exactly like Eq. (3.5.14) as described in the case of the asymptotic regression.

The source codes in FORTRAN of subroutine **REGNON** for both the asymptotic and logistic regressions and of the main program calling this subroutine are listed in Appendices A.3.5.2 and A.3.5.9, respectively.

Polynomial Regression

A polynomial equation of k-th order can be expressed as

$$y = b_0 + b_1x + b_2x^2 + b_3x^3 + \dots + b_kx^k \quad 3.5.21$$

If Eq. (3.5.21) is compared with Eq. (3.5.6) for the multilinear expression, it becomes obvious that the same regression procedure can be applied here with the following substitutions in the matrix representation (3.5.8):

$$x_1 \text{ by } x, x_2 \text{ by } x^2, x_3 \text{ by } x^3, \dots \text{ and } x_k \text{ by } x^k$$

With these substitutions, the matrix representation of the normal equations will be as follows:

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 & \dots & \sum x_i^k \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \dots & \sum x_i^{k+1} \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \dots & \sum x_i^{k+2} \\ \dots & \dots & \dots & \dots & \dots \\ \sum x_i^k & \sum x_i^{k+1} & \sum x_i^{k+2} & \dots & \sum x_i^{2k} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \dots \\ b_k \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \\ \dots \\ \sum x_i^k y_i \end{bmatrix}$$

3.5.22

This matrix representation (3.5.22) of the simultaneous equations for the polynomial regression can be solved for the unknown regression coefficients in the same way Eqs. (3.5.8) were solved in the case of multilinear regression.

When matrix **[A]** on the left hand side of matrix representation (3.5.22) is closely examined, it becomes obvious that it is very poorly conditioned. The number of equations that can be solved and thus the degree of the approximating polynomial is severely limited in most cases by roundoff error. A value of $k = 7$ or 8 will usually produce meaningless results on a 16-bit computer and with single precision arithmetic.

This problem is solved by dividing the array of the independent variable by the largest value of the array and thus making the largest number to be unity. The same technique is applied on the array with small numbers. Once the largest element is searched, and its value ($= X_{\max}$) is known, the regression coefficients, B_i , are corrected as follows :

$$B_i = \frac{B_i}{(X_{\max})^{i-1}}$$

where the subscript i varies from 1 to $k+1$.

The concept described above to avoid roundoff error due to formation of large numbers along the diagonal of the **[A]** matrix is original.

The source codes in FORTRAN of subroutine **REGPOL** for the polynomial regression and of the main program calling this subroutine is listed in Appendix A.3.5.10.

Multiple Correlation Coefficient, R

One criterion that is commonly used to illustrate the adequacy of a fitted regression model is the **coefficient of multiple determination, R^2** . The square root of the coefficient of multiple determination is called the **multiple correlation coefficient, R**. This quantity merely indicates what proportion of the total variation in the response independent variable is explained by the fitted model. This criterion is equally applicable for the model with single or multiple independent variable(s). It is expressed as

$$R = \sqrt{\frac{n \sum_{j=0}^k b_j \cdot g_j - \left(\sum_{i=1}^n Y_i \right)^2}{n \sum_{i=1}^n y_i^2 - \left(\sum_{i=1}^n Y_i \right)^2}} = \sqrt{\frac{n \sum_{j=0}^k b_j \cdot g_j - \frac{\left(\sum_{i=1}^n Y_i \right)^2}{n}}{\sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

where the arithmetic mean of all the Y-values,

$$\bar{Y} = \frac{1}{n} \sum_{i=1}^n Y_i$$

Standard Error

Standard error is defined as the square root of the mean error sum of squares and is expressed by

$$\text{Standard error} = \sqrt{\frac{\sum_{i=1}^n (y_i - Y_i)^2}{n}}$$

and

$$\text{Percent error} = \frac{\text{Standard error}}{\frac{\sum_{i=1}^n y_i}{n}} \cdot 100$$

where

Y : Experimental response

y : Model response.

References

1. **Daniel, C. and Gorman, J. W.**, *Fitting Equations to Computer Analysis of Multifactor Data*. John Wiley & Sons, New York, 458pp., 1980.
2. **Draper, N. and Smith, H.**, *Applied Regression Analysis*. John Wiley & Sons, New York, 709pp., 1981.
3. **Hornbeck, R. W.**, *Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ, 310pp., 1975.
4. **Snedecor, G. M. and Cochran, W. G.**, *Statistical Methods*. The Iowa State University Press, Ames, Seventh edition, 507pp., 1980.
5. **Walpole, R. E. and Myers, R. H.**, *Probability and Statistics for Engineers and Scientists*. Macmillan Publishing Co., New York and Collier Macmilan, London. Fifth edition, 766pp., 1993.
6. **Wylie, C.R., Jr. and L. C. Barrett**, *Advanced Engineering Mathematics*. McGraw-Hill Book Co., New York. Fifth edition, 1103pp., 1982.

CHAPTER 3.6

Solution of Initial and Boundary Value Problems

In the field of engineering in general, many physical relations appear in the form of differential equations. The derivation of differential equations from given physical situations is the essence of mathematical modeling. The difference between an ordinary algebraic equation and an ordinary differential equation lies in the fact that, in the former, the unknown is an ordinary variable in its original nonderivative form whereas in the latter the unknown is a function, and derivatives of the function appear in the equation.

An ordinary differential equation (ODE) shows the involvement of one or more derivatives of an unspecified function y of x , where the relation may also involve the dependent variable y itself, functions of independent variable x , and constants. The term **ordinary** separates it from the **partial** differential equation (PDE) where the dependent variable in its derivative form is a simultaneous function of more than one dependent variables. An ordinary differential equation may be of any order. It is said to be of order n if the n -th derivative of the dependent variable with respect to the independent one is the highest derivative in that equation.

A second-order differential equation is said to be linear if it can be written in the following form:

$$y'' + f(x)y' + g(x)y = r(x)$$

where y is the dependent variable, and x is the independent variable. When the right hand side of the equation is zero, that is, $r(x) = 0$, the equation is homogeneous, and when it is nonzero, the equation is called nonhomogeneous. Thus

$$y'' + 4y = e^{-x}\sin(x)$$

is a nonhomogeneous linear second-order differential equation, whereas

$$x^2y'' + xy' + (x^2 - 1)y = 0$$

is homogeneous.

Any differential equation of the second-order that cannot be expressed by the form shown above is nonlinear. The nonlinear differential equations contain terms having product of dependent variable in its differential and/or nondifferential form. The following equations are nonlinear:

$$y''y + y' = 0$$

and

$$(y'')^2 - (y')^2 = 1$$

We shall now identify two broad categories by which all problems involving ordinary differential equations can be classified. These are

- (i) Initial value problems and
- (ii) Boundary value problems.

A typical initial value problem is

$$Ay'' + By' + Cy = g(t)$$

with initial conditions, $y(t=0) = y_0$ and $y'(t=0) = v_0$

A typical example of a boundary value problem is

$$y'' + Dy' + Ey = h(x)$$

with boundary conditions, $y(x=0) = y_0$ and $y(x=L) = y_L$

Thus an initial value problem consists of a differential equation, which may be of any order, and initial condition(s), the number of which depends on the order/degree of the differential equation. The initial conditions, if more than one, should all be known at a single value of the independent variable.

The methods to find exact analytical solutions of ordinary differential equations vary from equation to equation and are dependent on the boundary and initial conditions. Even for veteran mathematicians, these methods are often complicated. There are many situations where an exact solution virtually does not exist due to nonlinearity in the differential equation together with complex boundary or initial conditions. The numerical approach to the solution of differential equation, on the other hand, does not require modifications to tackle nonlinearities in the differential equations and

complex boundary or initial conditions. Nevertheless, as in all other situations, the accuracy of the approximate solutions by numerical methods, unless verified by some means or other, such as experimental values, previous experience, etc., cannot be properly estimated and there is an inherent danger in accepting such solutions without a very vigorous and thorough scrutiny.

Initial value problems

Any initial value problem can be represented by one, or a set of more than one, coupled first-order differential equation(s) (DE), each with an initial condition. Thus an n -th-order DE can be split up into n first-order DEs. For example, the second-order DE to express the helical motion of particle inside a cyclone separator, expressed by

$$r'' + a.r' - b.r^3 = 0$$

can be split into two first-order DEs introducing another variable z to express the first derivative r' and transforming the original second-order DE into a first-order DE in z as follows:

$$\begin{aligned} r' &= z \text{ with initial condition, } r(t=t_0) = r_0 \text{ and} \\ z' &= r'' = -a.r' + b.r^3 \text{ with } r'(t=t_0) = z_0 \end{aligned}$$

Since any initial value problem can be expressed as a set of first-order ordinary differential equations, the numerical methods to solve first-order DEs will be of primary importance. Let us first consider a single first-order ordinary differential equation in the form:

$$y' = f(t, y) \quad \text{with } y(t=t_0) = y_0$$

The assumption here is that the function is analytic around a range of finite values of the independent variable. In finding a solution of such an equation, a stepwise approach is taken where the solution cannot be expected in the form of an equation as in the case of an analytic solution. Instead, the solution starts at a known value given by the initial condition, $y(t=t_0) = y_0$ and continues thereafter at selected values of the independent variable, t , usually equally spaced, until the solution covers the required range.

Euler method

The Taylor Series expansion of $y(t+h)$ is given by:

$$y(t+h) = y(t) + hy'(t) + \frac{h^2 y''(t)}{2!} + \frac{h^3 y'''(t)}{3!} + \frac{h^4 y^{iv}(t)}{4!} + \dots \quad 3.6.1$$

Now, neglecting the terms having second and higher derivatives of y ,

$$\begin{aligned} y(t+h) &= y(t) + h.y'(t) \\ &= y(t) + h.f(t,y) \end{aligned}$$

and, in general,

$$y_{n+1} = y_n + h.f(t_n, y_n) \quad 3.6.2$$

For small values of h , as the power of h is increased, the value of the terms neglected above is further decreased, and the accuracy of estimation of y is increased. If the first time through, the right hand side of Eq. (3.6.2) is evaluated by the initial condition ($y_n = y_0$ at $t_n = t_0$), the left hand side will denote the value of y at $t = t_0 + h$. When these new values of y and t are substituted on the right hand side of the same equation, it will calculate a new value of y at $t = t_0 + 2h$. This process is repeated until the new value of t reaches the desired point. This is called the **Euler method** or **Euler-Cauchy method**.

It is a first-order method because, in Eq. (3.6.2) only up to the first power of h is considered. The omission of the rest of the terms causes an error called the **truncation error**. There will be another type of error known as **round off error**, which is cumulative in nature, and it increases as the steps progress. In practice, this method is almost never used. Its importance lies in its simplicity and the insight that it offers to help understand the basic idea of such stepwise methods.

Example 3.6.1. Euler method:

Solve the following initial value problem by the Euler method to calculate y at $t = 0.2$:

$$y' = -2ty, \text{ with } h = 0.1 \text{ and initial condition } y(0) = 1$$

Solution:

Selecting $h = 0.1$, and starting with $n = 0$, the first value of y at $t = 0 + 0.1$ is calculated using Eq. (3.6.2) as

$$y_1 = y_0 + 0.1 * f(t_0, y_0)$$

where the values of y_0 and t_0 are obtained from the initial condition given and $f(t_0, y_0)$ is calculated from the original equation

$$f(t_n, y_n) = y'_n = -2t_n * y_n$$

Thus,

$$y_1 = 1.0 + 0.1(-2.0*0.0*1.0) \\ = 1.0$$

Now, taking this value of $y_1 = 1.0$ and corresponding $t = 0.1$, y_2 at $t = 0.1 + 0.1$ is calculated by Eq. (3.6.2) as

$$y_2 = 1.0 + 0.1(-2.0*0.1*1.0) \\ = 0.98$$

The exact values of y at $t=0.1$ and $t=0.2$ are 0.99004983 and 0.96078944, respectively, as obtained from the analytical solution,

$$y = e^{-t^2}$$

Improved Euler method

In Eq. (3.6.1), if the second derivative in y is substituted by $f'(t, y)$ which is expressed by the following difference equation,

$$f'(t, y) = \frac{f(t+h, y) - f(t, y)}{h}, \quad \text{we get} \\ y(t+h) = y(t) + h \cdot f(t, y) + h \frac{[f(t+h, y) - f(t, y)]}{2} \\ = y(t) + \frac{h}{2} [f(t+h, y) + f(t, y)]$$

Though for simplicity only the letter y has been used in the expressions for functions, such as $f(t+h, y)$ and $f(t, y)$, it should be noted that they are actually the values at steps $t+h$ and t , respectively.

In general, the above expression can be written as

$$y_{n+1} = y_n + \frac{h}{2} [f(t_{n+1}, y_{n+1}) + f(t_n, y_n)] \quad 3.6.3$$

In this case the truncation error is of the order of h^3 , and the method is called **Improved Euler method**. As observed from Eq. (3.6.3), the value of the function is to be evaluated at a value of y one step ahead of the initial condition. Thus by knowing only the initial condition, this formula cannot be used. In fact, in order to use this expression, y_{n+1} and consequently $f(t_{n+1}, y_{n+1})$ should first be calculated by the Euler method.

This type of method is called the **Predictor-Corrector method**, where one self-starting procedure is used to predict the value or values of the dependent variable at certain step(s) using the initial condition, and then a corrector formula is used to improve the accuracy of the variable at the same step location. In the present case, the Euler formula is used as the predictor and the Improved Euler formula as the corrector. This is a second-order method, thus having higher accuracy than the previous one.

Thus, in general terms, the predictor and the corrector of the improved Euler method can be written as

Predictor

$$y_{n+1}^* = y_n + f(t_n, y_n) \quad 3.6.4$$

Corrector

$$y_{n+1} = y_n + \frac{h}{2} [f(t_{n+1}, y_{n+1}^*) + f(t_n, y_n)] \quad 3.6.5$$

Example 3.6.2. Improved Euler method:

Let us solve the same problem given in Example 3.6.1.

Solution:

From the last solution, the predictor (3.6.4) gives the value of y_1^* as 1.0. Now using the corrector (3.6.5),

$$\begin{aligned} y_1 &= 1.0 + 0.05(-2.0 \cdot 0.1 \cdot 1.0 - 2.0 \cdot 0.0 \cdot 1.0) \\ &= 0.9900 \quad (\text{exact} = 0.99004983) \end{aligned}$$

For the next step the predictor is given by

$$\begin{aligned} y_2^* &= 0.99 + 0.1(-2.0 \cdot 0.1 \cdot 0.99) \\ &= 0.9702 \end{aligned}$$

and the corrector gives

$$y_2 = 0.99 + 0.05(-2.0 \cdot (0.1 + 0.1) \cdot 0.9702 - 2.0 \cdot 0.1 \cdot 0.99)$$

$$= 0.960696 \quad (\text{exact} = 0.96078944)$$

When compared to the exact values, the improved Euler method produces accuracy up to three decimal places, which is expected from a second-order method using a step size of 0.1.

Runge-Kutta method

The expression for the Euler method (Eq. 3.6.2) for $n = 0$ can be written as

$$\begin{aligned} y_1 &= y_0 + h.f(t_0, y_0) \\ &= y_0 + h. \left. \frac{dy}{dt} \right|_{t_0, y_0} \\ &= y_0 + \Delta y_1 \end{aligned} \quad 3.6.6$$

where

$$\Delta y_1 = h.f(t_0, y_0)$$

Now, $\frac{dy}{dt}$ is evaluated at $t_0 + \frac{h}{2}$ and $y_0 + \frac{h.f(t_0, y_0)}{2}$ to give

$$\begin{aligned} y_1 &= y_0 + h.f \left[t_0 + \frac{h}{2}, y_0 + \frac{h.f(t_0, y_0)}{2} \right] \\ &= y_0 + h.f \left[t_0 + \frac{h}{2}, y_0 + \frac{y_1}{2} \right] \\ &= y_0 + \Delta y_2 \end{aligned}$$

where

$$\Delta y_2 = h.f \left[t_0 + \frac{h}{2}, y_0 + \frac{y_1}{2} \right]$$

Thus the simplest second-order formula is

$$y_{n+1} = y_n + \frac{h}{2} f(t_{n+1/2}, y_{n+1/2}^*)$$

where

$$y_{n+1/2}^* = y_n + \frac{h}{2} f(t_n, y_n)$$

and

$$t_{n+1/2} = t_n + \frac{h}{2}$$

Generally, three or more estimates of Δy are computed and then a linear combination of all these estimates is finally used to determine y_{n+1} . The multiple evaluation of the function at different values of t and y adds accuracy to this procedure. Also, it can be noted that it is not a multistep formula, meaning that it does not require a value of y other than the initial one. Thus it is self-starting. The method based on this principle is known as the **Runge-Kutta method** named after German mathematicians Carl Runge (1856–1927) and Wilhelm Kutta (1867–1944). This method is available with various accuracies, starting from second-order. The most commonly used one is the fourth-order method, where the estimates of Δy expressed by k_1 , k_2 , k_3 , and k_4 are as follows:

$$k_1 = h.f(t_n, y_n) \quad 3.6.7.a$$

$$k_2 = h.f\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \quad 3.6.7.b$$

$$k_3 = h.f\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \quad 3.6.7.c$$

$$k_4 = h.f(t_n + h, y_n + k_3) \quad 3.6.7.d$$

Finally,

$$y_{n+1} = y_o + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4) \quad 3.6.7$$

Formulas of the Runge-Kutta type are among the most widely used for the numerical solution of ordinary differential equations due to their following advantages:

- (i) self starting,
- (ii) simple for programming,
- (iii) stable, and
- (iv) step size variation during the solution of a problem is possible.

On the other hand, the two main disadvantages of such formulas are as follows:

- (i) they require more computer time than other methods of comparable accuracy, and
- (ii) the estimate of local error at each step is difficult.

The error estimate for this method is not straightforward. To ensure accurate results, it is necessary to use a small step size. The optimum step size varies from problem to problem and one way to decide is to examine the

results with varying step sizes. The largest one offering the required accuracy should be selected. The estimate of truncation error per step is a better way of selecting an optimum step size. It is done by starting a solution for a selected interval by one step size and repeating the same with a step size half of the previous one. If the value of the solution at a point with the original step size is denoted by y_{n+1} , and that with the half step size to arrive at the same point by y_{n+1}^* , the truncation error between the interval, e_{n+1} is expressed by

$$e_{n+1} = \frac{y_{n+1}^* - y_{n+1}}{2^k - 1}$$

where k is the order of the method. Usually k is equal to four. The step size should be adjusted so that the error estimate does not exceed a predetermined small value in the range of 10^{-10} . The above error expression is obtained from the Richardson Extrapolation formula, the derivation of which is given in Chapter 3.1.

Example 3.6.3. Fourth-order Runge-Kutta method:

Let us solve the same problem given in Example 3.6.1.

Solution:

From the equation set (3.6.7), values of $k_1 \dots k_4$ are calculated using the initial conditions as follows:

$$\begin{aligned} k_1 &= 0.1(-2.0*0.0*1.0) \\ &= 0.0 \\ k_2 &= 0.1\left(-2.0*\left(0.0 + \frac{0.1}{2}\right)*\left(1.0 + \frac{0.0}{2}\right)\right) \\ &= -0.01 \\ k_3 &= 0.1\left(-2.0*\left(0.0 + \frac{0.1}{2}\right)*\left(1.0 + \frac{-0.01}{2}\right)\right) \\ &= -0.00995 \\ k_4 &= 0.1(-2.0*(0.0+0.1)*(1.0+(-0.00995))) \\ &= -0.01981 \end{aligned}$$

$$\begin{aligned} \text{and } y_1 &= 1.0 + \frac{1}{6} (0.0+2.0*(-0.01)+2.0*(-0.00995)+(-0.01981)) \\ &= 0.99004833 \text{ (exact} = 0.99004983) \end{aligned}$$

To calculate the next step, y_2 , again $k_1 \dots k_4$ are computed using the value of y_1 at a step location of t_0+h , that is, $(0.0+0.1)$:

$$\begin{aligned}
k_1 &= 0.1(-2.0*0.1*0.99004833) \\
&= -0.01980097 \\
k_2 &= 0.1\left(-2.0*\left(0.1+\frac{0.1}{2}\right)*\left(0.9900483+\frac{-0.01980097}{2}\right)\right) \\
&= -0.02940444 \\
k_3 &= 0.1\left(-2.0*\left(0.1+\frac{0.1}{2}\right)*\left(0.9900483+\frac{-0.02940444}{2}\right)\right) \\
&= -0.02926038 \\
k_4 &= 0.1(-2.0*(0.1+0.1)*(0.9900483+(-0.02926038))) \\
&= -0.03843152 \\
\text{and } y_2 &= 0.9900483 + \frac{1}{6}(-0.01980097+2.0*(-0.02940444) \\
&\quad + (2.0*(-0.02926038) + (-0.03843152))) \\
&= 0.96078798 \text{ (exact} = 0.96078944)
\end{aligned}$$

In the above example, the fourth-order Runge-Kutta method gives an accuracy up to five decimal places as compared to the exact values of y_1 and y_2 .

Predictor-Corrector methods

It has been observed that the Runge-Kutta method is simple and accurate but it consumes a considerable computing time. On the other hand, predictor-corrector methods, often called P-C methods, reduce computing time without sacrificing the desired accuracy. These are multistep methods and are easy to derive from the first principles, that is, Taylor Series expansion.

The expression for the predictor is often called the open formula and that for the corrector as closed formula. The open formula itself is sufficient to solve an ordinary differential equation. The name "closed" arrives from the fact that it is not as explicit as the open formula — it requires an iterative scheme for convergence. A predictor-corrector method employs first a predictor, and the predicted value is then corrected by a corrector that is iterated to convergence, reducing the actual error considerably as compared to the predictor.

The order of accuracy for the predictor should be equal to that for the corrector. Generally fourth-order formulas are used. The characteristics of open as well as closed formulas are such that these methods cannot be started without the help of other self starting methods. We shall examine these characteristics as these formulas will be derived.

There are many multistep formulas. The most commonly used ones are the **Adams-Bashforth open formula** of fourth-order and the **Adams-Moulton closed formula** of the same order. The derivations of them from the first principles follow.

Derivation of an open formula of fourth-order

Let us consider the Taylor Series expansion of $y(t+h)$, where it is necessary for h (which is an increment in t) to bear a small value and the function,

$$f(t,y) = \frac{dy}{dt} = y'(t) \quad 3.6.8$$

to be analytic in the solution domain, that is, the function should have a finite value in the range used for t .

$$y_{(t+h)}^* = y(t) + h \cdot y'(t) + \frac{h^2}{2!} \cdot y''(t) + \frac{h^3}{3!} \cdot y'''(t) + \frac{h^4}{4!} \cdot y^{iv}(t) + \frac{h^5}{5!} \cdot y^v(t) \quad 3.6.9$$

Now, combining Eq. (3.6.8) with Eq. (3.6.9),

$$y_{(t+h)}^* = y(t) + h \cdot f(t,y) + \frac{h^2}{2!} \cdot f'(t,y) + \frac{h^3}{3!} \cdot f''(t,y) + \frac{h^4}{4!} \cdot f'''(t,y) + \frac{h^5}{5!} \cdot f^{iv}(t,y) + \dots \quad 3.6.10$$

The backward finite difference expressions for the first, second, and the third derivatives of the function can be had from Chapter 3.1, which are as follows:

$$f'(t,y) = \frac{f(t,y) - f(t-h,y)}{h} + \frac{h}{2} \cdot f''(t,y) - \frac{h^2}{6} \cdot f'''(t,y) + \frac{h^3}{24} \cdot f^{iv}(t,y) - \dots$$

$$f''(t,y) = \frac{f(t,y) - 2f(t-h,y) + f(t-2h,y)}{h^2} + h \cdot f'''(t,y) - \frac{7h^2}{12} \cdot f^{iv}(t,y) + \dots$$

and

$$f'''(t,y) = \frac{f(t,y) - 3f(t-h,y) + 3f(t-2h,y) - f(t-3h,y)}{h^3} + \frac{3h}{2} \cdot f^{iv}(t,y) - \dots$$

Substituting these expressions of function derivatives in Eq.(3.6.10), the open formula of fourth-order is obtained as

$$y_{(t+h)}^* = y(t) + \frac{h}{24} [55f(t,y) - 59f(t-h,y) + 37f(t-2h,y) - 9f(t-3h,y)] \\ + \frac{251h^5}{720} \cdot f^{iv}(t,y) - \dots \quad 3.6.11$$

Neglecting the term with h^5 , the final expression of the predictor becomes

$$y_{n+1}^* = y_n + \frac{h}{24} (55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \quad 3.6.12$$

Since the terms neglected start with the fifth power of h , this is a fourth-order formula and is known as **Adams-Bashforth open formula or predictor**.

Derivation of a closed formula of fourth-order

The procedure here is similar to that used to derive the open formula (Eq. 3.6.12). In this case a backward Taylor Series expansion of $y(t)$ yields

$$y(t) = y\{(t+h)-h\} \\ = y(t+h) - h \cdot y'(t+h) + \frac{h^2}{2!} \cdot y''(t+h) - \frac{h^3}{3!} \cdot y'''(t+h) + \frac{h^4}{4!} \cdot y^{iv}(t+h) - \dots$$

Rearranging it and combining with Eq. (3.6.8),

$$y(t+h) = y(t) + h \cdot f(t+h,y) - \frac{h^2}{2!} \cdot f''(t+h,y) + \frac{h^3}{6!} \cdot f'''(t+h,y) - \frac{h^4}{24!} \cdot f^{iv}(t+h,y) \\ + \frac{h^5}{120!} \cdot f^{iv}(t+h,y) - \dots \quad 3.6.13$$

The backward finite difference expressions for the first, second, and the third derivatives of the function can be had from Chapter 3.1, which are as follows:

$$f'(t+h,y) = \frac{f(t+h,y) - f(t,y)}{h} + \frac{h}{2} \cdot f''(t+h,y) - \frac{h^2}{6} \cdot f'''(t+h,y) + \frac{h^3}{24} \cdot f^{iv}(t+h,y) - \dots \\ f''(t+h,y) = \frac{f(t+h,y) - 2f(t,y) + f(t-2,y)}{h^2} + h \cdot f'''(t+h,y) - \frac{7h^2}{12} \cdot f^{iv}(t+h,y) + \dots$$

and

$$f^{(4)}(t+h, y) = \frac{f(t+h, y) - 3f(t, y) + 3f(t-h, y) - f(t-2h, y)}{h^3} + \frac{3h}{2} \cdot f^{(iv)}(t+h, y) - \dots$$

Substituting these expressions of function derivatives in Eq.(3.6.13), the closed formula of fourth-order is obtained as

$$y(t+h) = y(t) + \frac{h}{24} [9f^*(t+h, y) + 19f(t, y) - 5f(t-h, y) + f(t-2h, y)] - \frac{19h^5}{720} \cdot f^{(iv)}(t+h, y) + \dots \quad 3.6.14$$

Neglecting the term with h^5 , the final expression of the corrector becomes,

$$y_{n+1} = y_n + \frac{h}{24} (9f_{n+1}^* + 19f_n - 5f_{n-1} + f_{n-2}) \quad 3.6.15$$

Since neglected terms start with the fifth power of h , this is also a fourth-order formula just like the open one. Here the function f_{n+1}^* can only be evaluated when the value of y_{n+1}^* is known by the open formula. This expression is known as **Adams-Moulton closed formula or the corrector**.

Estimation of truncation error

Let us denote the value of y_{n+1}^* as well as y_{n+1} without truncation as Y_{n+1} . Thus from Eqs. (3.6.11) and (3.6.12),

$$Y_{n+1} = y_{n+1}^* + \frac{251h^5}{720} \cdot f_n^{(iv)}$$

or

$$Y_{n+1} - y_{n+1}^* = \frac{251h^5}{720} \cdot f_n^{(iv)} \quad 3.6.16$$

and from Eqs. (3.6.14) and (3.6.15),

$$Y_{n+1} = y_n + \frac{h}{24} [9f(t+h, Y_{n+1}) + 19f_n - 5f_{n-1} + f_{n-2}] - \frac{19h^5}{720} \cdot f_{n+1}^{(iv)}$$

Now, subtracting Eq. (3.6.15) from the above yields

$$Y_{n+1} - y_{n+1} = \frac{9h}{24} [f(t+h, Y_{n+1}) - f(t+h, y_{n+1})] - \frac{19h^5}{720} \cdot f_{n+1}^{(iv)} \quad 3.6.17$$

Using the Mean-Value Theorem,

$$f(t_{n+1}, Y_{n+1}) - f(t_{n+1}, y_{n+1}) = (Y_{n+1} - y_{n+1}) \cdot \frac{df}{dy}(t_{n+1}, \xi_{n+1})$$

where ξ_{n+1} lies between Y_{n+1} and y_{n+1}

For a sufficiently small value of h ,

$$\frac{9h}{24} \cdot \left| \frac{df}{dy}(t_{n+1}, \xi_{n+1}) \right| \ll 1$$

and hence the first right hand term of Eq. (3.6.17) is negligible. Also the fourth derivatives f_n^{iv} and f_{n+1}^{iv} do not vary too much in the region of t between t_{n-3} and t_{n+1} . Thus

$$f_n^{iv} = f_{n+1}^{iv} = C$$

Utilizing the above information, Eqs. (3.6.16) and (3.6.17) become

$$Y_{n+1} - y_{n+1}^* = \frac{251h^5}{720} \cdot C \quad 3.6.18$$

and

$$Y_{n+1} - y_{n+1} = -\frac{19h^5}{720} \cdot C \quad 3.6.19$$

From Eqs. (3.6.18) and (3.6.19),

$$C \cdot h^5 = \frac{720}{270} (y_{n+1} - y_{n+1}^*)$$

Substituting this expression in Eq. (3.6.19), truncation error per step, e_{n+1} is obtained as

$$e_{n+1} = Y_{n+1} - y_{n+1} = -\frac{19}{270} \cdot (y_{n+1} - y_{n+1}^*) \quad 3.6.20$$

Similar open and closed formulas of fourth-order given by **Hamming** are as follows:

Hamming's Open formula

$$y_{n+1}^* = y_{n-3} + \frac{4h}{3} (2f_n - f_{n-1} + 2f_{n-2}) \quad 3.6.21$$

and

Hamming's Closed formula

$$y_{n+1} = \frac{9y_n - y_{n-2}}{8} + \frac{3.h}{8} (f_{n+1}^* + 2f_n - f_{n-1}) \quad 3.6.22$$

The truncation error per step for this procedure can be estimated by:

$$e_{n+1} = \frac{9}{121} (y_{n+1} - y_{n+1}^*) \quad 3.6.23$$

The closed formula has the advantage of being accurate, but, as an iterative procedure, it consumes more computer time. This drawback can be overcome if a reasonable guess of the dependent variable at the recent point is made, which will reduce the number of iterations considerably. In predictor-corrector methods, this is precisely what is done using the open formula. It has been observed that the number of iterations remains within 2 to reach a difference of 10^{-6} between the two consecutive iterated values given by the corrector when a reasonably small value of step size is chosen.

The predicted value can be improved by combining the error series of both the predictor and the corrector and introducing a modifier after the predictor.

Modifier for the Adams Formula

The truncation error term in the predictor is obtained from Eq. (3.6.11) as $(251h^5.C/720)$. Substituting in the expression just derived for $C.h^5$, this term becomes

$$\frac{251}{270} (y_{n+1} - y_{n+1}^*)$$

Thus its modifier is expressed by,

$$y_{n+1}^* = y_{n+1}^* + \frac{251}{270} (y_n - y_n^*) \quad 3.6.24$$

Similarly, for Hamming's P-C method, the modifier is given as

$$y_{n+1}^* = y_{n+1}^* + \frac{112}{121} (y_n - y_n^*) \quad 3.6.25$$

The left hand side of the modifier equation is the modified predicted value, and the same term on the right hand side is the unmodified predicted value. The use of such a modifier helps in reducing the number of iterations

required by the corrector for convergence. Since Eqs. type (3.6.24) and (3.6.25) use both predictor and corrector values, the modifiers can only be calculated after these values are known the first time through.

The fourth-order Runge-Kutta method requires four calculations of the function at each step, whereas any P-C method requires one calculation of the function by the predictor and one evaluation for each iteration by the corrector. If the convergence criterion is reached within two iterations, which is the usual case, the P-C method still saves at least one computation of the function at each step. This is quite substantial when a small step size is chosen and the desired range to cover is wide.

As has been discussed earlier, these predictor-corrector methods require additional information besides the initial condition. In Eq. (3.6.12), if f_{n-3} is calculated at an initial condition, that is, at y_0 , the values of y_1 , y_2 , and y_3 are still required to calculate y_4^* . Now, these previous values should be calculated with comparable accuracy by a self-starting method. In fact, the fourth-order Runge-Kutta method is universally used for this purpose.

Example 3.6.4. Adams Predictor-Corrector method:

Considering the same problem given in Example 3.6.1., y values at the first three steps are calculated by the fourth-order Runge-Kutta method (excluding the initial condition, which is $y_0 = 1.0$ at $t=0.0$) as

$$y_1 = 0.99004983; y_2 = 0.96078944 \text{ and } y_3 = 0.91393117$$

The first four functions are calculated using the corresponding y -values as follows:

$$\begin{aligned} f_0 &= -2.0 \cdot 0.0 \cdot 1.0 = 0.0 \\ f_1 &= -2.0 \cdot 0.1 \cdot 0.99004983 = -0.19800997 \\ f_2 &= -2.0 \cdot 0.2 \cdot 0.96078944 = -0.38431578 \\ f_3 &= -2.0 \cdot 0.3 \cdot 0.91393117 = -0.54835870 \end{aligned}$$

Now, substituting these values in the predictor (3.6.12),

$$\begin{aligned} y_4^* &= 0.91393117 + \frac{0.1}{24.0} (55.0 \cdot (-0.54835870) - 59.0 \cdot (-0.38431578) + \\ &\quad 37.0 \cdot (-0.19800997) - 9.0 \cdot 0.0) \\ &= 0.85221673 \end{aligned}$$

The function, f_4^* is now calculated using this predicted value of y at a new step value of $0.3+0.1$:

$$\begin{aligned} f_4^* &= -2.0 \cdot 0.4 \cdot 0.85221673 \\ &= -0.68177338 \end{aligned}$$

This value is used by the corrector (3.6.15) along with the previous three f values, namely, f_3 , f_2 , and f_1 to calculate the final value of y_4 ,

$$\begin{aligned} y_4 &= 0.91393117 + \frac{0.1}{24.0} (9.0 \cdot (-0.68177338) + 19.0 \cdot (-0.54835870) - \\ &\quad 5.0 \cdot (-0.38431578) + (-0.19800997)) \\ &= 0.852134475 \end{aligned}$$

For the first iteration, this corrected value of y_4 is used to compute the new value of f_4^* as

$$\begin{aligned} f_4^* &= -2.0 \cdot 0.4 \cdot 0.852134475 \\ &= -0.68170758 \end{aligned}$$

and the corrected value of y_4 is again calculated by the same Eq. (3.6.15) as 0.85213687 (exact value : 0.85214379).

A comparison showing the estimation of y values by various methods described in this text is provided in Table 3.6.1 using the same differential equation with a step size of 0.01. The accuracy of such an estimation given by each method, besides being dependent on the step size, also varies with the type of the differential equation.

The Runge-Kutta formulas are the most dependable ones, and, when the behavior of the differential equation(s) is not very certain, these methods are the ones to be used.

In case of a stiff differential equation, when there are different scales of the independent variable, and the dependent variable is behaving in a irregular fashion (oscillating manner), the step size should be reduced. Generally, when the solution of the differential equation is of the exponential type, such as $y = a + be^{ct}$, such instability arises.

Following the discussion on different methods of solving initial value problems involving first-order differential equations, a program in FORTRAN is developed. Its source code is listed in Appendix A.3.6.1. It will be a good exercise to follow each step of this program. Sufficient comments are provided to make this task easy.

The program has been divided into two parts — 1. main program, where the input data are provided by the user; and 2. subroutine **ODE(KK,J,F)** to solve DE by five different methods, e.g., Euler's (KK=1), Improved Euler's (KK=2), Runge-Kutta's (KK=3), Adams' P-C (KK=4), and Hamming's P-C (KK=5); and 3. the function, which the user has to provide at the end of the program. The example problem has been solved by this program, and the

results are tabulated in Table 3.6.1. Results can be printed at any required intervals.

Table 3.6.1. A comparison of results by various methods.

Step location	Y v a l u e s					
	Euler method	Improved Euler	R-K method	Adam's P-C	Hamming's P-C	Exact
0.0	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
0.1	0.99103472	0.99004979	0.99004983	0.99004983	0.99004983	0.99004983
0.2	0.96266529	0.96078939	0.96078944	0.96078944	0.96078944	0.96078944
0.3	0.91651980	0.91393130	0.91393119	0.91393118	0.91393118	0.91393119
0.4	0.85520640	0.85214435	0.85214379	0.85214379	0.85214379	0.85214379
0.5	0.78206883	0.77880222	0.77880078	0.77880078	0.77880078	0.77880078
0.6	0.70088419	0.69767915	0.69767633	0.69767632	0.69767632	0.69767633
0.7	0.61554109	0.61263112	0.61262639	0.61262639	0.61262639	0.61262639
0.8	0.52973601	0.52729948	0.52729242	0.52729242	0.52729242	0.52729242
0.9	0.44672033	0.44486770	0.44485807	0.44485806	0.44485806	0.44485807
1.0	0.36912013	0.36789167	0.36787944	0.36787944	0.36787944	0.36787944

The variable to denote the initial condition has been used as Y0, and the dependent and the independent variables in the function **F(T,YY)** as YY and T, respectively. The input data that the program asks are : NVALUE, TIVALU, FVALUE and DELT, that is, number of values to be shown or stored in output file **DATA.OUT**, initial and final values of the independent variable and the step size, respectively. The use of Eqs. (3.6.20) and (3.6.23) for the estimation of truncation error and (3.6.24) and (3.6.25) for predictor modification has been demonstrated in this program. The output shows the Y values with corresponding step locations, total number of steps, accumulative truncation error, and the step size. A double precision accuracy is recommended for these multistep solution methods.

Higher-order initial value problems

So far, only the first-order differential equations have been considered. The solution of higher-order DEs does not require special methods other than the ones just discussed. In fact, all the higher-order methods such as fourth-order Runge-Kutta and fourth-order predictor-correctors (Adams and Hamming's) are used to solve higher order initial and boundary value problems. Any n-th order differential equation can be split up into n first-order differential equations, and coupled sets of any order can thus be expressed as a coupled set of first-order equations. Thus,

$$\frac{d^n y}{dt^n} = f(t,y,y',y'', \dots ,y^{n-1}) \qquad 3.6.26$$

can be split up as

$$\frac{dy}{dt} = z_1; \quad \frac{dy'}{dt} = z_2; \quad \frac{dy''}{dt} = z_3; \quad \dots \quad \frac{dy^{n-2}}{dt} = z_{n-1}$$

subject to initial conditions

$$y(t_0) = y_{0,0}; \quad y'(t_0) = y_{1,0}; \quad y''(t_0) = y_{2,0}; \quad \dots \quad y^{n-1}(t_0) = y_{n-1,0}$$

Now, Eq. (3.6.26) is rewritten as

$$\frac{d^n y}{dt^n} = f(t, y, z_1, z_2, \dots, z_{n-1})$$

For a better understanding of the solution of a higher-order DE, and, also, the necessity that any predictor-corrector method require a same order self-starting method, the procedure to solve higher-order DE by the fourth-order Runge-Kutta method alone will be discussed first. The incorporation of any P-C method then becomes a matter of choice or convenience.

The Runge-Kutta formulas for the coupled first-order DEs can be expressed in sequence as

$$\begin{aligned} D_1 y_1 &= h \cdot z_1(t) \\ D_1 y_2 &= h \cdot z_2(t) \\ D_1 y_3 &= h \cdot z_3(t) \\ &\dots\dots\dots \\ D_1 y_{n-1} &= h \cdot z_{n-1}(t) \\ D_1 y_n &= h \cdot f(t, y(t), z_1(t), z_2(t), \dots, z_{n-1}(t)) \end{aligned}$$

$$\begin{aligned} D_2 y_1 &= h \cdot (z_1(t) + D_1 y_2/2) \\ &= D_1 y_1 + h \cdot D_1 y_2/2 \\ D_2 y_2 &= h \cdot (z_2(t) + D_1 y_3/2) \\ &= D_1 y_2 + h \cdot D_1 y_3/2 \\ D_2 y_3 &= h \cdot (z_3(t) + D_1 y_4/2) \\ &= D_1 y_3 + h \cdot D_1 y_4/2 \\ &\dots\dots\dots \\ D_2 y_{n-1} &= h \cdot (z_{n-1}(t) + D_1 y_n/2) \\ &= D_1 y_{n-1} + h \cdot D_1 y_n/2 \\ D_2 y_n &= h \cdot f(t+h/2, y(t) + D_1 y_1/2, z_1(t) + D_1 y_2/2, \dots, z_{n-1}(t) + D_1 y_{n-1}/2) \end{aligned}$$

$$\begin{aligned} D_3 y_1 &= h \cdot (z_1(t) + D_2 y_2/2) \\ &= D_1 y_1 + h \cdot D_2 y_2/2 \\ D_3 y_2 &= h \cdot (z_2(t) + D_2 y_3/2) \end{aligned}$$

$$\begin{aligned}
 &= D_1 y_2 + h \cdot D_2 y_3 / 2 \\
 D_3 y_3 &= h \cdot (z_3(t) + D_2 y_4 / 2) \\
 &= D_1 y_3 + h \cdot D_2 y_4 / 2 \\
 &\dots\dots\dots \\
 D_3 y_{n-1} &= h \cdot (z_{n-1}(t) + D_2 y_n / 2) \\
 &= D_1 y_{n-1} + h \cdot D_2 y_n / 2 \\
 D_3 y_n &= h \cdot f(t+h/2, y(t)+D_2 y_1/2, z_1(t)+D_2 y_2/2, \dots, z_{n-1}(t)+D_2 y_{n-1}/2) \\
 &\dots\dots\dots \\
 D_4 y_1 &= h \cdot (z_1(t) + D_3 y_2) \\
 &= D_1 y_1 + h \cdot D_3 y_2 \\
 D_4 y_2 &= h \cdot (z_2(t) + D_3 y_3) \\
 &= D_1 y_2 + h \cdot D_3 y_3 \\
 D_4 y_3 &= h \cdot (z_3(t) + D_3 y_4) \\
 &= D_1 y_3 + h \cdot D_3 y_4 \\
 &\dots\dots\dots \\
 D_4 y_{n-1} &= h \cdot (z_{n-1}(t) + D_3 y_n) \\
 &= D_1 y_{n-1} + h \cdot D_3 y_n \\
 D_4 y_n &= h \cdot f(t+h, y(t)+D_3 y_1, z_1(t)+D_3 y_2, \dots, z_{n-1}(t)+D_3 y_{n-1})
 \end{aligned}$$

Now, the expressions of y , z_1 , z_2 , z_3 , $\dots$, z_{n-1} for the next step (that is, $t = t+h$) are given by:

$$\begin{aligned}
 y(t+1) &= y(t) + \frac{1}{6} (D_1 y_1 + 2D_2 y_1 + 2D_3 y_1 + D_4 y_1) \\
 z_1(t+1) &= z_1(t) + \frac{1}{6} (D_1 y_2 + 2D_2 y_2 + 2D_3 y_2 + D_4 y_2) \\
 z_2(t+1) &= z_2(t) + \frac{1}{6} (D_1 y_3 + 2D_2 y_3 + 2D_3 y_3 + D_4 y_3) \\
 &\dots\dots\dots \\
 z_{n-1}(t+1) &= z_{n-1}(t) + \frac{1}{6} (D_1 y_n + 2D_2 y_n + 2D_3 y_n + D_4 y_n)
 \end{aligned}$$

This process continues with an increment of h in each step until the upper limit of the independent variable t is reached.

Before attempting to write a program on the procedure just developed, we will develop other procedures to solve (i) n first-order DEs and (ii) n higher-order DEs. The reason for doing this is to develop a general program capable of handling any number of DEs of any order simultaneously. This may sound complicated at this stage, but, after development of the other two procedures, it will be rather obvious to sense the similarities among them and, based on the common points, the general program can be developed.

Sets of coupled first-order differential equations

Let us examine the following set of first-order DEs:

$$\begin{aligned}\frac{dy}{dt} &= f_1(t, y, z_1, z_2, \dots, z_{n-1}) \\ \frac{dz_1}{dt} &= f_2(t, y, z_1, z_2, \dots, z_{n-1}) \\ \frac{dz_2}{dt} &= f_3(t, y, z_1, z_2, \dots, z_{n-1}) \\ &\dots\dots\dots \\ \frac{dz_{n-1}}{dt} &= f_n(t, y, z_1, z_2, \dots, z_{n-1})\end{aligned}$$

with a proper initial condition for each.

The general Runge-Kutta formula for these equations are as follows:

$$\begin{aligned}D_1y_1 &= h.f_1(t, y(t), z_1(t), z_2(t), \dots, z_{n-1}(t)) \\ D_1y_2 &= h.f_2(t, y(t), z_1(t), z_2(t), \dots, z_{n-1}(t)) \\ D_1y_3 &= h.f_3(t, y(t), z_1(t), z_2(t), \dots, z_{n-1}(t)) \\ &\dots\dots\dots \\ D_1y_n &= h.f_n(t, y(t), z_1(t), z_2(t), \dots, z_{n-1}(t)) \\ \\ D_2y_1 &= h.f_1(t+h/2, y(t)+D_1y_1/2, z_1(t)+D_1y_2/2, \dots, z_{n-1}(t)+D_1y_{n-1}/2) \\ D_2y_2 &= h.f_2(t+h/2, y(t)+D_1y_1/2, z_1(t)+D_1y_2/2, \dots, z_{n-1}(t)+D_1y_{n-1}/2) \\ D_2y_3 &= h.f_3(t+h/2, y(t)+D_1y_1/2, z_1(t)+D_1y_2/2, \dots, z_{n-1}(t)+D_1y_{n-1}/2) \\ &\dots\dots\dots \\ D_2y_n &= h.f_n(t+h/2, y(t)+D_1y_1/2, z_1(t)+D_1y_2/2, \dots, z_{n-1}(t)+D_1y_{n-1}/2) \\ \\ D_3y_1 &= h.f_1(t+h/2, y(t)+D_2y_1/2, z_1(t)+D_2y_2/2, \dots, z_{n-1}(t)+D_2y_{n-1}/2) \\ D_3y_2 &= h.f_2(t+h/2, y(t)+D_2y_1/2, z_1(t)+D_2y_2/2, \dots, z_{n-1}(t)+D_2y_{n-1}/2) \\ D_3y_3 &= h.f_3(t+h/2, y(t)+D_2y_1/2, z_1(t)+D_2y_2/2, \dots, z_{n-1}(t)+D_2y_{n-1}/2) \\ &\dots\dots\dots \\ D_3y_n &= h.f_n(t+h/2, y(t)+D_2y_1/2, z_1(t)+D_2y_2/2, \dots, z_{n-1}(t)+D_2y_{n-1}/2) \\ \\ D_4y_1 &= h.f_1(t+h, y(t)+D_3y_1, z_1(t)+D_3y_2, \dots, z_{n-1}(t)+D_3y_{n-1}) \\ D_4y_2 &= h.f_2(t+h, y(t)+D_3y_1, z_1(t)+D_3y_2, \dots, z_{n-1}(t)+D_3y_{n-1}) \\ D_4y_3 &= h.f_3(t+h, y(t)+D_3y_1, z_1(t)+D_3y_2, \dots, z_{n-1}(t)+D_3y_{n-1}) \\ &\dots\dots\dots \\ D_4y_n &= h.f_n(t+h, y(t)+D_3y_1, z_1(t)+D_3y_2, \dots, z_{n-1}(t)+D_3y_{n-1})\end{aligned}$$

Now, the expressions of $y, z_1, z_2, z_3, \dots, z_{n-1}$ for the next step (that is, $t = t+h$) are given by:

$$\begin{aligned}
y(t+1) &= y(t) + \frac{1}{6} (D_1y_1 + 2D_2y_1 + 2D_3y_1 + D_4y_1) \\
z_1(t+1) &= z_1(t) + \frac{1}{6} (D_1y_2 + 2D_2y_2 + 2D_3y_2 + D_4y_2) \\
z_2(t+1) &= z_2(t) + \frac{1}{6} (D_1y_3 + 2D_2y_3 + 2D_3y_3 + D_4y_3) \\
&\dots\dots\dots \\
z_{n-1}(t+1) &= z_{n-1}(t) + \frac{1}{6} (D_1y_n + 2D_2y_n + 2D_3y_n + D_4y_n)
\end{aligned}$$

Sets of coupled higher-order differential equations

For simplicity, first we will consider n second-order differential equations. The procedure later can be generalized for n m -th order DEs. Let us express n second-order equations as

$$\begin{aligned}
\frac{d^2y}{dt^2} &= f_1(t, y, z_{11}, z_1, z_{21}, z_2, z_{31}, \dots, z_{n-1}, z_{n1}) \\
\frac{d^2z_1}{dt^2} &= f_2(t, y, z_{11}, z_1, z_{21}, z_2, z_{31}, \dots, z_{n-1}, z_{n1}) \\
\frac{d^2z_2}{dt^2} &= f_3(t, y, z_{11}, z_1, z_{21}, z_2, z_{31}, \dots, z_{n-1}, z_{n1}) \\
&\dots\dots\dots \\
\frac{d^2z_{n-1}}{dt^2} &= f_n(t, y, z_{11}, z_1, z_{21}, z_2, z_{31}, \dots, z_{n-1}, z_{n1})
\end{aligned}$$

with two initial conditions for each.

Now, expressing each second-order equation as two first-order DEs,

$$\begin{aligned}
\frac{dy}{dt} &= z_{11} \\
\frac{dz_{11}}{dt} &= f_1(t, y, z_{11}, z_1, z_{21}, z_2, z_{31}, \dots, z_{n-1}, z_{n1}) \\
\frac{dz_1}{dt} &= z_{21} \\
\frac{dz_{21}}{dt} &= f_2(t, y, z_{11}, z_1, z_{21}, z_2, z_{31}, \dots, z_{n-1}, z_{n1}) \\
\frac{dz_2}{dt} &= z_{31} \\
\frac{dz_{31}}{dt} &= f_3(t, y, z_{11}, z_1, z_{21}, z_2, z_{31}, \dots, z_{n-1}, z_{n1}) \\
&\dots\dots\dots
\end{aligned}$$

$$\frac{dz_{n-1}}{dt} = z_{n1}$$

$$\frac{dz_{n1}}{dt} = f_n(t, y, z_{11}, z_1, z_{21}, z_2, z_{31}, \dots, z_{n-1}, z_{n1})$$

The Runge-Kutta formulas for solving $y, z_{11}, z_1, z_{21}, \dots, z_{n-1}$ and z_{n1} are as follows:

$$\begin{aligned} D_1 y_1 &= h \cdot z_{11}(t) \\ D_1 y_{11} &= h \cdot f_1(t, y(t), z_{11}(t), z_1(t), z_{21}(t), z_2(t), z_{31}(t), \dots, z_{n-1}(t), z_{n1}(t)) \\ D_1 y_2 &= h \cdot z_{21}(t) \\ D_1 y_{21} &= h \cdot f_2(t, y(t), z_{11}(t), z_1(t), z_{21}(t), z_2(t), z_{31}(t), \dots, z_{n-1}(t), z_{n1}(t)) \\ &\dots \\ D_1 y_n &= h \cdot z_{n1}(t) \\ D_1 y_{n1} &= h \cdot f_n(t, y(t), z_{11}(t), z_1(t), z_{21}(t), z_2(t), z_{31}(t), \dots, z_{n-1}(t), z_{n1}(t)) \\ &\dots \\ D_2 y_1 &= h \cdot (z_{11}(t) + D_1 y_1 / 2) \\ D_2 y_{11} &= h \cdot f_1(t+h/2, y(t) + D_1 y_1 / 2, z_{11}(t) + D_1 y_{11} / 2, \dots, z_{n-1}(t) + D_1 y_n / 2, z_{n1}(t) + D_1 y_{n1} / 2) \\ D_2 y_2 &= h \cdot (z_{21}(t) + D_1 y_{21} / 2) \\ D_2 y_{21} &= h \cdot f_2(t+h/2, y(t) + D_1 y_1 / 2, z_{11}(t) + D_1 y_{11} / 2, \dots, z_{n-1}(t) + D_1 y_n / 2, z_{n1}(t) + D_1 y_{n1} / 2) \\ &\dots \\ D_2 y_n &= h \cdot (z_{n1}(t) + D_1 y_{n1} / 2) \\ D_2 y_{n1} &= h \cdot f_n(t+h/2, y(t) + D_1 y_1 / 2, z_{11}(t) + D_1 y_{11} / 2, \dots, z_{n-1}(t) + D_1 y_n / 2, z_{n1}(t) + D_1 y_{n1} / 2) \\ D_3 y_1 &= h \cdot (z_{11}(t) + D_2 y_1 / 2) \\ D_3 y_{11} &= h \cdot f_1(t+h/2, y(t) + D_2 y_1 / 2, z_{11}(t) + D_2 y_{11} / 2, \dots, z_{n-1}(t) + D_2 y_n / 2, z_{n1}(t) + D_2 y_{n1} / 2) \\ D_3 y_2 &= h \cdot (z_{21}(t) + D_2 y_{21} / 2) \\ D_3 y_{21} &= h \cdot f_2(t+h/2, y(t) + D_2 y_1 / 2, z_{11}(t) + D_2 y_{11} / 2, \dots, z_{n-1}(t) + D_2 y_n / 2, z_{n1}(t) + D_2 y_{n1} / 2) \\ &\dots \\ D_3 y_n &= h \cdot (z_{n1}(t) + D_2 y_{n1} / 2) \\ D_3 y_{n1} &= h \cdot f_n(t+h/2, y(t) + D_2 y_1 / 2, z_{11}(t) + D_2 y_{11} / 2, \dots, z_{n-1}(t) + D_2 y_n / 2, z_{n1}(t) + D_2 y_{n1} / 2) \\ &\dots \\ D_4 y_1 &= h \cdot (z_{11}(t) + D_3 y_1 / 2) \\ D_4 y_{11} &= h \cdot f_1(t+h, y(t) + D_3 y_1 / 2, z_{11}(t) + D_3 y_{11} / 2, \dots, z_{n-1}(t) + D_3 y_n / 2, z_{n1}(t) + D_3 y_{n1} / 2) \\ D_4 y_2 &= h \cdot (z_{21}(t) + D_3 y_{21} / 2) \\ D_4 y_{21} &= h \cdot f_2(t+h, y(t) + D_3 y_1 / 2, z_{11}(t) + D_3 y_{11} / 2, \dots, z_{n-1}(t) + D_3 y_n / 2, z_{n1}(t) + D_3 y_{n1} / 2) \\ &\dots \\ D_4 y_n &= h \cdot (z_{n1}(t) + D_3 y_{n1} / 2) \\ D_4 y_{n1} &= h \cdot f_n(t+h, y(t) + D_3 y_1 / 2, z_{11}(t) + D_3 y_{11} / 2, \dots, z_{n-1}(t) + D_3 y_n / 2, z_{n1}(t) + D_3 y_{n1} / 2) \end{aligned}$$

Now, the expressions of $y, z_{11}, z_1, z_{21}, \dots, z_{n-1}, z_{n1}$ for the next step (that is, $t = t+h$) are given by:

$$y(t+1) = y(t) + \frac{1}{6} (D_1 y_1 + 2D_2 y_1 + 2D_3 y_1 + D_4 y_1)$$

$$z_{11}(t+1) = z_{11}(t) + \frac{1}{6} (D_1y_{11} + 2D_2y_{11} + 2D_3y_{11} + D_4y_{11})$$

$$z_1(t+1) = z_1(t) + \frac{1}{6} (D_1y_2 + 2D_2y_2 + 2D_3y_2 + D_4y_2)$$

$$z_{21}(t+1) = z_{21}(t) + \frac{1}{6} (D_1y_{21} + 2D_2y_{21} + 2D_3y_{21} + D_4y_{21})$$

$$z_{n-1}(t+1) = z_{n-1}(t) + \frac{1}{6} (D_1y_n + 2D_2y_n + 2D_3y_n + D_4y_n)$$

$$z_{n1}(t+1) = z_{n1}(t) + \frac{1}{6} (D_1y_{n1} + 2D_2y_{n1} + 2D_3y_{n1} + D_4y_{n1})$$

The Runge-Kutta step identifiers D_1y , D_2y , D_3y , and D_4y for higher-order, more than one first-order and more than one higher-order DEs are in fact similar to k_1 , k_2 , k_3 , and k_4 , respectively for a single first-order DE. These sometimes define functions of just one variable, and the other time functions of more than one variables. A close observation of these identifiers reveals that, even in the case of single variable functions, e.g., $D_1y_1 = h.z_1(t)$ for one higher-order DE, and $D_1y_1 = h.z_{11}(t)$ for a set of higher-order DEs, etc., these can safely be expressed as

$$D_1y_1 = h.f(t, y, z_1, z_2, \dots, z_{n-1})$$

and

$$D_1y_1 = h.f(t, y(t), z_{11}(t), z_1(t), z_{21}(t), z_2(t), z_{31}(t), \dots, z_{n-1}(t), z_{n1}(t))$$

without altering their values.

The same thing can be said about the higher terms in D , such as D_2y_1 , D_3y_2 , etc. Thus all the procedures discussed so far can be cast into a common procedure given for solving n ordinary differential equations. This fact is the key point in developing a general program/subroutine to solve any number of any order ordinary differential equations simultaneously. The FORTRAN source codes of a subroutine **RKPC(KK,J,F)** having such general capability are listed in Appendix A.3.6.5. Another subroutine **START** provides starting values to **RKPC**. It is listed in Appendix A.3.6.4. The main program **N_ODE.FOR** calling these subroutines is listed in Appendix A.3.6.6.

In the program, the variable **Y(NEQ,N,NORDER)** represents y values when the number of first-order DEs, **NEQ** is 1, z_1 -values when **NEQ** is 2, ... z_{n-1} -values when **NEQ** is n . The variable **N** represents the specific step location. This is not allowed to exceed 5 to save memory in the case of the predictor-corrector methods, such as Adams, Hamming's, etc. The new values at the recent four locations are taken from the last four values among the five calculated at the preceding step. In the case of the Runge-Kutta method alone, only two values will suffice, but, in order to use the same subroutine for any of the three methods, e.g., Runge-Kutta, Adams P-C and Hamming's P-C, the maximum value of **N** has been kept equal to 5.

The initial conditions are provided through a separate external program named **DATA.FOR**, which is included in the main program by a **\$INCLUDE** metacommand. The first-order DEs are also supplied through another external program named **FCTN.FOR**.

The variable **NORDER** defines the number of higher-order equations. Thus for one higher-order DE, **Y** can be defined in two ways, e.g., for one fourth-order DE — (i) four first-order DEs, thus **NEQ** = 4 and **NORDER** = 1 [**Y**(4,**N**,1)]; and (ii) one fourth-order DE, where **NEQ** = 1 and **NORDER** = 4 [**Y**(1,**N**,4)]. The program accepts both, only inputs and functions should be defined differently.

An example will make this clear. Let us consider the following fourth-order differential equation:

$$y^{iv} = -4y.y''' \text{ [exact solution: } y = (t + 1)^{-1}]$$

with initial conditions

$$y(t_0=0) = 1$$

$$y'(t_0=0) = z_1(t_0=0) = -1$$

and

$$y''(t_0=0) = z_2(t_0=0) = 2$$

$$y'''(t_0=0) = z_3(t_0=0) = -6$$

In the program **DATA.FOR**, the initial conditions are given for the variable **Y0(NEQ,NORDER)** as follows:

(i) Four first-order DEs

$$Y0(1,1) = 1.0$$

$$Y0(2,1) = -1.0$$

$$Y0(3,1) = 2.0$$

$$Y0(4,1) = -6.0$$

and

(ii) One fourth-order DE

$$Y0(1,1) = 1.0$$

$$Y0(1,2) = -1.0$$

$$Y0(1,3) = 2.0$$

$$Y0(1,4) = -6.0$$

The file **FCTN.FOR** for function input defines the function as **F(I,KK,M,T)**, where **I** denotes the function number, **KK** is the step position that is obtained from the subroutine, and **M** is the order of the differential equation(s), which is supplied by the main program. **T** is the independent variable. The split up first-order functions are defined as follows:

(i) Four first-order DEs

```

IF(I .EQ. 1) THEN
  F = Y(2,KK,M)
ELSEIF(I .EQ. 2) THEN
  F = Y(3,KK,M)
ELSEIF(I .EQ. 3) THEN
  F = Y(4,KK,M)
ELSEIF(I .EQ. 4) THEN
  F = -4.0*Y(1,KK,M)*Y(4,KK,M)
ENDIF
RETURN
END

```

As was explained earlier, $Y(1, KK, M)$ represents y values, $Y(2, KK, M)$ are the z_1 -values (y' -values), $Y(3, KK, M)$ are the z_2 -values (y'' -values), and $Y(4, KK, M)$ are the z_3 -values (y''' -values). Here it should be carefully noted that if there is any trigonometric function, such as **sine**, **cosine**, **tangent**, etc., it should be written with double precision notation such as **DSIN**, **DCOS**, **DTAN**, etc. This is a necessity and without such appropriate double precision notations for some functions, the compiler will give an error message. Another two such common functions are **DABS** and **DSQRT**. When in doubt, Appendix A.2.1 should be consulted.

(ii) One fourth-order DE

```

IF(M .EQ. 1) THEN
  F = Y(I, KK, 2)
ELSEIF(M .EQ. 2) THEN
  F = Y(I, KK, 3)
ELSEIF(M .EQ. 3) THEN
  F = Y(I, KK, 4)
ELSEIF(M .EQ. 4) THEN
  F = -4.0*Y(I, KK, 1)*Y(I, KK, 4)
ENDIF
RETURN
END

```

In this case, the value of **I** is supplied to the **FUNCTION F** by the main program which is 1. The fourth-order DE is split up as follows:

For **M** = 1, **F** = y' ; for **M** = 2, **F** = y'' ; for **M** = 3, **F** = y''' ; and for **M** = 4, **F** is the original function, i.e., $y^{iv} = -4y.y'''$, where y is $Y(I, KK, 1)$, y' is $Y(I, KK, 2)$, y'' is $Y(I, KK, 3)$ and y''' is $Y(I, KK, 4)$.

With these inputs through **DATA.FOR** and **FCTN.FOR**, when the main program **N_ODE.FOR** listed in Appendix A.3.6.6 is compiled and executed,

it asks for a one digit number option among three fourth-order methods, namely, [1] for Runge-Kutta, [2] for Adams Predictor-Corrector, and [3] for Hamming's Predictor-Corrector. Once chosen, it asks for input data for (i) number of equation, (ii) order of equation, (iii) printing interval, (iv) initial and final values of the independent variable, and (v) step size. For the above example these are 4, 1, 10 (number of values to be printed can be 100 as maximum), 0.0, 1.0 (final value of the independent variable can be any other value greater than zero), and 0.01 (step size can vary, but should not be too large or too small. A range of 0.1 to 0.01 is appropriate for most of the problems).

The output shows solutions for y , z_1 , z_2 , etc. with corresponding step values, cumulative truncation error for the two P-C methods, and the total number of steps. The output is also saved in a file, created by the program, named **DATA.OUT**.

Boundary value problems

The minimum order of the differential equation for a boundary value problem should be two. There are various methods to deal with a boundary value problem, e.g., matrix method, shooting method, shooting and superposition method, and so on. In the matrix method, the nonlinearity of the differential equation is complex and poses a problem. The shooting and superposition method is a simple version of the shooting method itself and is applied when a set of linear ordinary differential equations is involved.

On the other hand, there are many established and efficient techniques to tackle initial value problems, as we have already experienced. The shooting method solves the differential equation exactly as an initial value problem employing powerful and accurate fourth-order methods, such as Runge-Kutta, Adams P-C, Hamming's P-C, etc. Since one of the conditions defines the boundary [the rest can be considered as initial condition(s)], an assumption has to be made to turn the boundary value problem to an equivalent initial value problem. This technique of transformation is precisely what a shooting method employs, thus offering the advantage of using the above mentioned powerful methods. The formerly developed subroutine **RKPC(KK,J,F)** will be used to solve a boundary value problem involving any order (greater than or equal to two) and any number of ordinary differential equation, which may be linear or nonlinear.

Shooting method for solution of a boundary value problem

Let us consider the previous typical example of a boundary value problem given at the beginning of this chapter:

$$y'' + Dy' + Ey = h(x) \quad 3.6.27$$

with boundary conditions, $y(x=x_0) = y_0$ and $y(x=L) = y_L$.

The same equation can be solved as an initial value problem if the first boundary condition is considered as the first initial condition, and the required second initial condition is defined as

$$\frac{dy}{dx} (x=x_0) = U$$

A very important fact should be emphasized here — that both the conditions should be defined at the same location, that is, at the same value of the independent variable. In this example it has been x_0 .

Now, let us examine the transformed problem. The value of U is not known, and it has been assumed to satisfy the actual boundary condition. Now, initially if an arbitrary value of U is assumed, and the equation is solved as an initial value problem taking a proper step size to reach exactly at the point $x = L$, the solution will give a value of y at $x = L$. There will be every chance, though, that this value will be far away from y_L , the actual given value at the boundary. What will be the next step then? Another value of U can be tried and another until the actual boundary condition is satisfied within a specified tolerance. So it becomes an iterative process and an appropriate and effective convergence criterion becomes a necessity to minimize the number of iterations for a fast convergence.

We see that the boundary condition is a direct function of our assumed value of U . In Chapter 3.2, powerful methods were discussed for calculation of the root(s) of a function. By definition, roots are zeros of a function, that is, the function should be made zero if its roots are to be calculated. This concept can effectively be employed here for a rapid convergence on U to yield the boundary condition within a given tolerance. The actual boundary condition can be expressed in the following functional form:

$$f(U) = y(x=L) - y_L = 0$$

Now it turns out to be a root solving problem where the objective is to find a value of U for which $f(U)$ becomes zero. Our experience with the Newton-Raphson method of root finding has been very positive, but it needs the analytic expression for the first derivative of the function. The function $f(U)$ in this case is not explicit, and thus the Newton-Raphson method cannot be applied directly. The other method that uses the same principle of the Newton-Raphson method but does not suffer the limitation on the availability of an analytic expression for the first derivative of the function should be the answer. It is the Secant method, and, in the present context, it can be rewritten as

$$U_{n+1} = U_n - \frac{f(U_n) \cdot (U_n - U_{n-1})}{[f(U_n) - f(U_{n-1})]}$$

In order to evaluate U_{n+1} , two values of U , namely, U_n and U_{n-1} should be known *a priori*.

Thus the strategy will be to provide two initial guesses of U , and, using these values one at a time, the transformed initial value problem should be solved for $y_n(x=L)$ and y_{n-1} and, consequently, $f(U_n)$ and $f(U_{n-1})$ to calculate a new value of U , i.e., U_{n+1} . This process is continued until the convergence criterion is reached, and, when it does we get the solution.

A source code listing of a program **BOUND.FOR** to solve boundary value problem is given in Appendix A.3.6.7. This program is capable of solving any order or number of differential equations, though the number of first-order ODEs or the degree of one ODE has been limited to 10 in it. It uses the same subroutine **RKPC(KK,J,F)** developed originally for solving initial value problems (listing in Appendix A.3.6.5). Unlike the initial value program, it does not accept the order of DE more than one in its input. Thus for one n -th order DE, input data can only be specified as if there were n first-order DEs.

The boundary conditions are entered through an external FORTRAN file **DATA.FOR**, and the differential equations through **FCTN.FOR** the same way it was done in the initial value problems. The only difference here is that the last boundary condition is not specified in the **DATA.FOR** file. Thus this file always has one condition less than the number or degree of ODE. The boundary condition is entered through a **READ** statement along with other values, such as, number of ODE, printing interval, initial and final values of independent variable, and step size.

Let us explain its functioning with an example. A third-order differential equation is chosen for this purpose:

$$y''' = -2y \cdot y'' + (y')^2 - 1$$

with

$$y(x=0) = 0; y'(x=0) = 0 \text{ and } y'(x=4.4) = 1$$

The equivalent initial value problem should have the following initial condition

$$y(x=0) = 0; y'(x=0) = 0 \text{ and } y''(x=0) = U$$

Now, the task is to find a value of U for which $y'(x=4.4) - 1$ should be close to zero, or in other words, less than or equal to a tolerance value supplied by the program. This value has been given as 10^{-6} in the program. The inputs are provided by two external files as follows:

In DATA.FOR

$$Y0(1,1) = 0.0$$

$$Y0(2,1) = 0.0$$
In FCTN.FOR

```

IF(I .EQ. 1) THEN
  F = Y(2, KK, M)
ELSEIF(I .EQ. 2) THEN
  F = Y(3, KK, M)
ELSEIF(I .EQ. 3) THEN
  F = -2.0*Y(1, KK, M)*Y(3, KK, M)+Y(2, KK, M)**2-1.0
ENDIF
RETURN
END

```

With these files written, when the program **BOUND.FOR** is compiled and executed, it asks for a one digit number option among three fourth-order methods, namely, [1] for Runge-Kutta, [2] for Adams Predictor-Corrector and [3] for Hamming's Predictor-Corrector. Once selected, it asks for input data: (i) number of equation, (ii) initial value, (iii) final value of the independent variable, and (iv) the value at the boundary. For the above example these are 3, 0.0, 4.4 and 1.0. After this the program shows the number of divisions for a step size of 0.1. This is to help users decide about the next question, which is about its value along with the value of the printing interval (number of values to be printed can be a maximum of 100). For a step size of 0.1, the number of divisions is given as 44 and the printing interval as 4. Step size can vary but should not be too large or too small. A range of 0.1 to 0.01 is appropriate for most of the problems. Then the two initial guesses for U , when asked, have been given as 1.0 and 2.0. If these values are given too wildly, the program may give a "**Real Math Overflow**" message. If this happens, it is advised to try with new limits on U .

The output shows solutions for y , z_1 , z_2 , etc., with corresponding step values, absolute error between given and calculated $f(U)$, and the total number of iterations. The output is also saved in a file, created by the program, named **DATA.OUT**. With two guesses of U as 1.0 and 2.0, the number of iterations for the above problem was 5 to converge with an absolute error of $0.14088689 \times 10^{-06}$.

The program has been developed in such a manner that it automatically takes care of the last initial condition when this last boundary condition is given on the derivative one degree less than that in the last equivalent initial condition. One example or two will clarify this. In the present example, the degree on the derivative of the equivalent initial condition [$y''(0) = U$] is two,

and the degree on the derivative of the last boundary condition [$y'(4.4) = 1$] is one, which is two minus one. In a second-order equation (3.6.24), the equivalent initial condition is a first-order derivative and the last boundary condition is a zero-order, which also falls into the logic of the program.

When this is not the case, e.g., if the last boundary condition in the above example is given on y instead of y' , as

$$y(x=0) = 0; y'(x=0) = 0 \text{ and } y(x=1.2) = 0.6654$$

only two lines of the main program **BOUND.FOR** should be modified replacing **NEQ-1** with **NEQ-2** as follows:

$$Y1(1) = Y(\text{NEQ-2}, L, 1) - \text{XBOND}$$

and

$$Y1(2) = Y(\text{NEQ-2}, L, 1) - \text{XBOND}$$

In fact, $Y(\text{NEQ}, L, M)$ represents y -, y' -, y'' -, ..., y^{n-1} - values depending upon the values of **NEQ**. When **NEQ** is 1 it represents y , when it is 2, it represents y' and so on.

In **BOUND.FOR**, the subroutine **RKPC** is called to calculate each step one at a time. Another program **BOUND1.FOR** is developed where subroutines **START1** and **RKPC1(PRNT,F)** calculate all the steps when they are called just once. The results are printed at required interval by another small subroutine **PRNT(INTV1)**. The source codes of **PRNT.FOR**, **START1.FOR**, **RKPC1.FOR**, and **BOUND1.FOR** are listed in Appendices A.3.6.8 through A.3.6.11.

There is one more important aspect to discuss before closing this chapter. Many times the last boundary condition is specified at an infinite value of the independent variable. Since a real number cannot assume an infinite value, such problems should be solved with an assumed upper boundary value. Considering the convergence of the method with this value, another value should be chosen, which may be greater than the first one, and the problem should be solved again. This process should continue until the converged method produces results that are very close to each other. Let this be explained by the following example:

$$y''' = -y \cdot y''$$

with

$$y(0) = 0, y'(0) = 0 \text{ and } y'(\infty) = 2$$

First, let us give a finite value of 3.0 and 5.0 to infinity and solve this problem with a step size of 0.1. The first two guesses of U are given as 1.0 and 5.0. The results with these input values are shown in Table 3.6.1.

Table 3.6.1. Results from the solution of example problem.

x	y	y'	y	y'
	L = 3.0		L = 5.0	
0.0	0.00000000	0.00000000	0.00000000	0.00000000
0.5	0.16582555	0.66056407	0.16557250	0.65956020
1.0	0.65100118	1.26136310	0.65002571	1.25953130
1.5	1.39881980	1.69431110	1.39680980	1.69208810
2.0	2.30887830	1.91324930	2.30574840	1.91103500
2.5	3.28748500	1.98518860	3.28327590	1.98308190
3.0	4.28487020	2.00000000	4.27962310	1.99794460
3.5			5.27924080	1.99984290
4.0			6.27921540	1.99999250
4.5			7.27921440	1.99999980
5.0			8.27921430	2.00000000

The absolute error on y' using the boundary value as 3.0 was calculated as $0.12930479 \times 10^{-9}$, whereas with 5.0 it was $0.76844531 \times 10^{-10}$. In both cases, it required six iterations for convergence. Thus after comparison between these two results, it is concluded that the value 5.0 effectively represents infinity for this particular problem.

References

1. **Burden, R. L. and Faires, J. D.**, *Numerical Analysis*. Fifth edition, Prindle, Weber & Schmidt, Boston, 1993.
2. **Dorn, W. S. and McCracken, D. D.**, *Numerical Methods with FORTRAN IV Case Studies*. John Wiley & Sons, New York, 1972.
3. **Hamming, R.W.**, *Numerical Methods for Scientists and Engineers*. Second Edition, International Student Edition, McGraw-Hill, Inc., Tokyo, Japan, 1973.
4. **Hildebrand, F. B.**, *Introduction to Numerical Analysis*. Second Edition, Dover Publications, New York, 1987.
5. **Hornbeck, R. W.**, *Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ, 310 pp., 1975.
6. **Kreyszig, E.**, *Advanced Engineering Mathematics*. Fourth edition, John Wiley & Sons, New York, 1979.
7. **Wylie, C. R., Jr. and L. C. Barrett**, *Advanced Engineering Mathematics*. Fifth edition, McGraw-Hill Book Co., New York, 1982.

CHAPTER 3.7

Solution of Partial Differential Equations

A partial differential equation (PDE) differs from an ordinary differential equation (ODE) in the fact that it describes the change in the dependent variable with respect to more than one independent variable in the system, e.g., time and position, whereas the latter describes this change with respect to only one independent variable. The mass, momentum, and energy balances made on the system yield such equations of change. The name *partial* comes from the fact that the dependent variable partially depends on time and partially on position.

The classification of the linear second-order PDEs is generally made by the form of the family of curves; namely elliptic, hyperbolic, and parabolic; that each of the equation represents. In most of the applications in food and agricultural engineering, the dependent variable is either temperature or moisture content. In general, let us denote it as U , instead of T (for temperature) or M (for moisture content). Thus according to the classification of PDE, these are as follows:

Elliptic

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2} = 0 \text{ or constant} \quad 3.7.1$$

Hyperbolic

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2} = \frac{\partial^2 U}{\partial t^2} \quad 3.7.2$$

Parabolic

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2} = \frac{\partial U}{\partial t} \quad 3.7.3$$

The above equations are in three-dimensional form for a slab geometry. When the right hand side of Eq. (3.7.1) is zero it is called Laplace's equation, and, when it is a constant, it is known as Poisson's equation. The hyperbolic equation represented by Eq. (3.7.2) is often called a wave equation, and the parabolic equation (3.7.3) is a heat or diffusion equation. The method of solution of hyperbolic PDE is different from that of the other two types.

However, in our later discussion, solutions of only one- and two-dimensional parabolic PDE for regular geometry will be considered because unsteady state (transient) heat conduction and mass diffusion equations are of this type. For isometric properties of the object under consideration and well-behaved initial and boundary conditions, one dimensional approximation becomes reasonably good. Persons interested in higher-order PDEs consisting of more than three independent variables will find it a straightforward extension of the simple forms presented here.

The derivation of the parabolic PDE in one-dimension is simple. If q is the rate of heat transfer per unit area perpendicular to its flow, the general form of the unsteady state heat transfer equation is given by

$$-\frac{\partial}{\partial x}(q \cdot A) = \rho \cdot C_p \cdot A \cdot \frac{\partial T}{\partial t} \quad 3.7.4$$

The right hand side of Eq. (3.7.4) is the gradient of the rate of sensible heat transfer, where ρ and C_p are the density and specific heat of the material in kg/m^3 and $\text{J/kg} \cdot ^\circ\text{C}$, respectively. For slab geometry, the area A is constant, that is, it does not change in the direction of heat flow and thus can be canceled. For a cylindrical or a spherical geometry, however, area is a function of the diameter, hence it cannot be canceled on the two sides of Eq. (3.7.4).

Using Fourier's law of heat conduction, the heat flux, q can be expressed as

$$q = -k \cdot \frac{\partial T}{\partial x} \quad 3.7.5$$

where k is the thermal conductivity in $\text{W/m} \cdot ^\circ\text{C}$.

Substituting the expression for q , into Eq. (3.7.4), and remembering that for a slab geometry, the area A does not come in to picture,

$$k \cdot \frac{\partial^2 T}{\partial x^2} = \rho \cdot C_p \cdot \frac{\partial T}{\partial t}$$

$$\frac{\partial T}{\partial t} = \frac{k}{\rho \cdot C_p} \cdot \frac{\partial^2 T}{\partial x^2}$$

$$\frac{\partial T}{\partial t} = \alpha \cdot \frac{\partial^2 T}{\partial x^2} \quad 3.7.6$$

where α is called **thermal diffusivity** (in m^2/s) and is expressed by:

$$\alpha = \frac{k}{\rho \cdot C_p}$$

In Eq. (3.7.3), the value of diffusivity has been taken as unity. It needs one condition in time and two in the boundaries to solve Eq. (3.7.6). The first is called the **initial condition** and the other two are called **boundary conditions**.

Approximation of the derivatives in Eq. (3.7.6) can be made by finite difference expressions as discussed in Chapter 3.1. Thus using forward difference for the left hand and central difference for the right hand sides of Eq. (3.7.6),

$$\frac{T_{i+1,j} - T_{i,j}}{\Delta t} = \alpha \cdot \frac{T_{i,j+1} - 2 \cdot T_{i,j} + T_{i,j-1}}{\Delta x^2} \quad 3.7.7$$

where the subscript **i** is used for a step in time and **j** for a step in space. Generally, the counter for time step **i** will stay between 1 and 2 and **j** between 2 and n, where n is the number of divisions along the principal dimension.

Solving the above equation for temperature at the next step in time, $T_{i+1,j}$

$$T_{i+1,j} = D \cdot T_{i,j-1} + (1 - 2 \cdot D) \cdot T_{i,j} + D \cdot T_{i,j+1} \quad 3.7.8$$

where

$$D = \frac{\alpha \cdot \Delta t}{\Delta x^2} \quad 3.7.9$$

In each time step, temperatures will be calculated by Eq. (3.7.8) for node points $j = 2$ up to n . The temperatures on the first ($j=1$) and the last ($j=n+1$) node points will be calculated by appropriate boundary conditions. The following grid notation shown in Figure 3.7.1 will be helpful in understanding the scheme just discussed. Due to the fact that the unknown in Eq. (3.7.8) can be explicitly determined, this is called an **explicit scheme**.

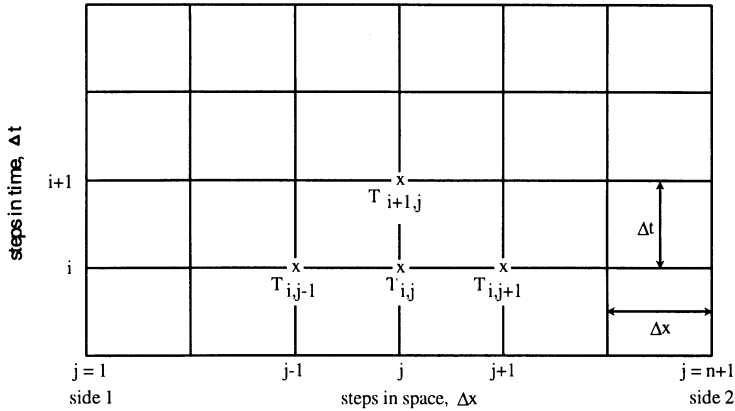


Figure 3.7.1. Finite difference grid notation in one dimension.

The explicit scheme suffers from being unstable or conditionally stable. If the center term in Eq. (3.7.8) is dropped, it becomes stable. Thus the obvious choice becomes

$$1 - 2.D = 0$$

that is, $D = 0.5$. Substituting this value into Eq. (3.7.9),

$$\Delta t = \frac{0.5 \cdot \Delta x^2}{\alpha} \quad 3.7.10$$

In fact, any value of time step less than or equal to the right hand side of Eq. (3.7.10), ensures the stability of the explicit scheme.

For any regular geometry, that is, slab, cylindrical, or spherical a general explicit scheme can be developed without canceling the area term in Eq. (3.7.4). From now on, for the sake of generality, the letter $\mathbf{r}$ will be used in place of $\mathbf{x}$ even when slab geometry is under consideration. Now the left hand side of Eq. (3.7.4) is expressed as

$$-\frac{\partial}{\partial r}(q.A) = -\frac{[q_{i,j+1/2} \cdot A_{j+1/2} - q_{i,j-1/2} \cdot A_{j-1/2}]}{\Delta r} \quad 3.7.11$$

The space notations $(j+1/2)$ and $(j-1/2)$ denote the midpoints between $\mathbf{j}+1$ and $\mathbf{j}$, and between $\mathbf{j}$ and $\mathbf{j}-1$, respectively. Thus

$$q_{i,j+1/2} = \frac{q_{i,j+1} + q_{i,j}}{2} \quad 3.7.12$$

$$A_{j+1/2} = \frac{A_{j+1} + A_j}{2} \quad 3.7.13$$

and

$$q_{i,j-1/2} = \frac{q_{i,j} + q_{i,j-1}}{2} \quad 3.7.14$$

$$A_{j-1/2} = \frac{A_j + A_{j-1}}{2} \quad 3.7.15$$

Now, using Eq. (3.7.5) in Eqs. (3.7.12) and (3.7.14) and substituting the final expressions in Eq. (3.7.11), Eq. (3.7.4) becomes:

$$\begin{aligned} -\frac{\partial}{\partial r}(q \cdot A) &= k \cdot \left[\frac{T_{i,j+1} - T_{i,j}}{\Delta r} \cdot \frac{A_{j+1} + A_j}{2} - \frac{T_{i,j} - T_{i,j-1}}{\Delta r} \cdot \frac{A_j + A_{j-1}}{2} \right] / \Delta r \\ &= \rho \cdot C_p \cdot A_j \cdot \frac{T_{i+1,j} - T_{i,j}}{\Delta t} \end{aligned}$$

In this derivation, the thermophysical properties, such as density (ρ in kg/m^3), specific heat (C_p in $\text{J/kg}^\circ\text{C}$), and thermal conductivity (k in $\text{W/m}^\circ\text{C}$) have been kept independent of space (x or r in m) and time (t in s). It is worth noting that, though SI units are used in this text, all the expressions are independent of the unit system. The only requirement is that the units should be consistent. The algebraic manipulation of the above equation gives,

$$T_{i+1,j} = T_{i,j} + E \cdot [(T_{i,j+1} - T_{i,j}) \cdot (A_{j+1} + A_j) - (T_{i,j} - T_{i,j-1}) \cdot (A_j + A_{j-1})] \quad 3.7.16$$

where

$$E = \frac{\alpha \cdot \Delta t}{2 \cdot \Delta r^2 \cdot A_j}$$

Generalization of the method for any shape

The program that will be written on the theory just discussed should be capable of taking care of any regular shape, e.g., plate/slab, cylinder, or sphere. It is done as follows:

Slab geometry

In Eq. (3.7.16), the area terms A_{j-1} , A_j , and A_{j+1} should be substituted by any constant. These are taken as unity here, that is,

$$A_{j-1} = A_j = A_{j+1} = 1.0 \quad 3.7.17$$

Cylindrical geometry

In Eq. (3.7.16), areas are expressed as

$$A_j = 2\pi \cdot [R - (j-1) \cdot \Delta r] \quad 3.7.18$$

Spherical geometry

In Eqs. (3.7.16),

$$A_j = 4\pi \cdot [R - (j-1) \cdot \Delta r]^2 \quad 3.7.19$$

where R is the radius (in m).

Initial and Boundary conditions

The solution of a partial differential equation is dependent on the initial and boundary conditions. Each type of condition yields unique solution, and there is a different way to arrive at the analytical solution that may exist. On the other hand, once the finite difference expression for the PDE is developed, it can easily be associated with different initial and boundary conditions and an approximate solution can be obtained.

Initially the object is considered at a constant temperature or the temperature may be position dependent (temperature as a function of position). The initial condition, thus is expressed as follows:

$$T(r, t=0) = T_0 \text{ or } f(r) \quad 3.7.20$$

Boundary conditions specify the temperature as a function of time at the two boundaries of the object and may be of different types. There are three principal categories of boundary conditions, **Dirichlet**, **Neumann**, and **Robbins**. An intermediate condition combining Dirichlet and Neumann conditions is called the **Cauchy condition**.

Dirichlet conditions

The dependent variable is expressed as a constant or as a function of time at the boundaries. An example would be a variable (time dependent) heat source on one side and a constant temperature maintained on the other side of an object.

Neumann conditions

The first derivative of the dependent variable with respect to position is given as a constant or as a function of time. When this derivative is zero, the side is perfectly insulated. These are also called **derivative boundary conditions**.

Robbins conditions

These conditions exist when the boundary is exposed to free convection - thus the heat flow by conduction is made equal to that by convection at the boundary. These are also known as **convective boundary conditions**.

Among the three conditions mentioned, the following will be discussed in detail:

- (i) one side at constant temperature, T_{c1} , and the other side is insulated (Cauchy condition). These are expressed as

at side 1:

$$T(r=0, t) = T_{c1} \quad 3.7.21$$

at side 2:

$$\frac{\partial T}{\partial r}(r = L, t) = 0$$

The condition at side 2 is called one form of the derivative boundary conditions (Neumann conditions). The general derivative condition is obtained by equating the temperature gradient, β , to some value, which may be a constant. In the case of an insulated boundary, rate of flow of heat across it is zero, that is, β gets a zero value. The general form of the derivative boundary condition is developed as follows:

At the left hand side:

$$\frac{\partial T}{\partial r} = \frac{T_{i,1} - T_{i,2}}{\Delta r} = \beta$$

$$\text{giving, } T_{i,1} = T_{i,2} + \beta \cdot \Delta r \quad 3.7.22$$

At the right hand side:

$$\frac{\partial T}{\partial r} = \frac{T_{i,n} - T_{i,n+1}}{\Delta r} = \beta$$

$$\text{giving, } T_{i,n+1} = T_{i,n} - \beta \cdot \Delta r \quad 3.7.23$$

This condition is also expressed by extending the boundary by one extra/imaginary node point, that is, 0-th for side 1 and (n+2)-th for side 2. This, however, cannot be used at the center of solid cylindrical and spherical geometries, which will be discussed in detail later.

Now, expressing $\frac{\partial T}{\partial r}$ by central difference form, for side 1,

$$\frac{T_{i,0} - T_{i,2}}{2 \cdot \Delta r} = \beta$$

Thus the imaginary temperature is given by

$$T_{i,0} = T_{i,2} + 2 \cdot \beta \cdot \Delta r \quad 3.7.24$$

Similarly, for side 2,

$$\frac{T_{i,n} - T_{i,n+2}}{2 \cdot \Delta r} = \beta$$

and the imaginary temperature,

$$T_{i,n+2} = T_{i,n} - 2 \cdot \beta \cdot \Delta r \quad 3.7.25$$

considering heat flow from side 1 towards side 2 in all the cases.

Eq. (3.7.16) with $j=1$ (side 1) gives

$$T_{i+1,1} = T_{i,1} + E \cdot [(T_{i,2} - T_{i,1})(A_2 + A_1) - (T_{i,1} - T_{i,0})(A_1 + A_0)]$$

Now, substituting $T_{i,0}$ by the expression in Eq. (3.7.24),

$$T_{i+1,1} = T_{i,1} + E \cdot [(T_{i,2} - T_{i,1})(A_2 + A_1) - (T_{i,1} - T_{i,2} - 2 \cdot \beta \cdot \Delta r)(A_1 + A_0)] \quad 3.7.26$$

The imaginary outside area A_0 does not pose any problem for slab geometry, where it can be taken as unity or a constant. In the case of a cylindrical or a spherical geometry, it should be calculated by adding one step in space with its outside radius ($r_1 + \Delta r$).

Similarly for $j = n + 1$ (side 2), Eq. (3.7.16) becomes

$$T_{i+1,n+1} = T_{i,n+1} + E \cdot [(T_{i,n+2} - T_{i,n+1})(A_{n+2} + A_{n+1}) - (T_{i,n+1} - T_{i,n})(A_{n+1} + A_n)]$$

Substituting $T_{i,n+2}$ by the expression in Eq. (3.7.25),

$$T_{i+1,n+1} = T_{i,n+1} + E \cdot [(T_{i,n} - 2\beta \cdot \Delta r - T_{i,n+1})(A_{n+2} + A_{n+1}) - (T_{i,n+1} - T_{i,n})(A_{n+1} + A_n)] \quad 3.7.27$$

The fictitious area, A_{n+2} , for a hollow cylinder or sphere can be calculated by the inner radius minus one step in space ($r_{n+1} - \Delta r$). It should be noted, however, that A_{n+1} for a solid cylinder or a sphere is zero, the $(n+1)$ -th node being at its center and A_{n+2} is equal to A_n .

(ii) both the sides are at constant temperatures, side 1 at T_{c1} and side 2 at T_{c2} (Dirichlet condition). This is expressed as

$$T(r=0,t) = T_{c1} \quad 3.7.28$$

$$\text{and, } T(r=L,t) = T_{c2} \quad 3.7.29$$

Now, if $T_{c1} = T_{c2}$, temperature distribution for one-half of the geometry should be the mirror image of the other half, hence only half the geometry should be solved, and, at the center, the derivative boundary condition exists, that is,

$$\frac{\partial T}{\partial r}(r = L/2, t) = 0$$

and Eq. (3.7.23) or Eq. (3.7.27) with $\beta = 0$ should apply.

(iii) one side is at T_{c1} and the other side is exposed to free convection (combination of Dirichlet and Robbins conditions). The first one is given by Eq. (3.7.21).

Temperature on the other side at any step in time can be computed as follows by balancing the rate of flow of heat by conduction between the last two nodes and that by convection from the last node.

Heat conducted between n -th and $(n+1)$ -th nodes, q_{cond} by Fourier's law is given by,

$$q_{\text{cond}} = -k \cdot A_{n+1/2} \frac{T_{i,n+1} - T_{i,n}}{\Delta r}$$

$$q_{\text{conv}} = h_2 \cdot A_{n+1} \cdot (T_{i,n+1} - T_{c2})$$

where h_2 is the heat transfer coefficient at the side 2 (in $\text{W/m}^2 \cdot ^\circ\text{C}$). Now, equating the above two equations, the temperature at the other boundary is given by

$$T_{i,n+1} = \frac{T_{c2} + DD2 \cdot T_{i,n}}{1 + DD2} \quad 3.7.30$$

where

$$DD2 = \frac{k \cdot (A_{n+1} + A_n)}{2A_{n+1} \cdot h_2 \cdot \Delta r} \quad 3.7.31$$

If, at side 1, a convective boundary condition exists, by similar balance

$$h_1 \cdot A_1 \cdot (T_{c1} - T_{i,1}) = k \cdot \frac{A_{1+1/2} \cdot (T_{i,1} - T_{i,2})}{\Delta r}$$

Finally, solving for $T_{i,1}$ yields

$$T_{i,1} = \frac{T_{c1} + DD1 \cdot T_{i,2}}{1 + DD1} \quad 3.7.32$$

where

$$DD1 = \frac{k \cdot (A_1 + A_2)}{2A_1 \cdot h_1 \cdot \Delta r} \quad 3.7.33$$

(iv) both the sides are exposed to free convection (Robbins conditions).

In this case the temperatures at side 1 and 2 are calculated as before and Eqs. (3.7.32) with (3.7.33) and Eqs. (3.7.30) with (3.7.31) apply. When $T_{c1} = T_{c2}$ and $h_1 = h_2$, again due to symmetry, only one-half of the geometry needs to be solved when one side is exposed to free convection and the other side (center) is with a derivative boundary condition. It often occurs when the entire object is exposed to a single environment.

Calculation of average temperature

The numerical method gives temperature values at each node point, which are the point values. More often it is required to know the average temperature of the object. The following method was used to calculate the average temperature of the product, T_m .

If the pertinent dimension of the object to heat transfer is divided into n parts (in the direction perpendicular to heat flow), the mass of individual parts, such divided, can be expressed as,

$$m_1 = V_1 \cdot \rho$$

$$m_2 = V_2 \cdot \rho$$

$$m_3 = V_3 \cdot \rho$$

$$\dots\dots\dots$$

$$m_n = V_n \cdot \rho$$

where $V_1, V_2, \dots, V_n$ are the volumes of each part, and ρ is the density, which was considered constant with respect to temperature.

The sensible heat flux in each part is

$$\begin{aligned} q_1 &= V_1 \cdot \rho \cdot C_p \cdot (T_{i,1+1/2} - T_r) \\ q_2 &= V_2 \cdot \rho \cdot C_p \cdot (T_{i,2+1/2} - T_r) \\ q_3 &= V_3 \cdot \rho \cdot C_p \cdot (T_{i,3+1/2} - T_r) \\ &\dots\dots\dots \\ q_n &= V_n \cdot \rho \cdot C_p \cdot (T_{i,n+1/2} - T_r) \end{aligned}$$

where

T_r : Reference temperature, $^{\circ}\text{C}$

C_p : Specific heat (considered constant with respect to temperature), $\text{J/kg} \cdot ^{\circ}\text{C}$.

The sum of the heat content of all the individual parts should be equal to the heat content of the entire object, which is

$$q = V \cdot \rho \cdot C_p \cdot (T_m - T_r)$$

Mathematically,

$$\sum_{j=1}^n \left[V_j \cdot (T_{i,j+1/2} - T_r) \cdot \rho \cdot C_p \right] = V \cdot \rho \cdot C_p \cdot (T_m - T_r) \quad 3.7.34$$

If the reference temperature, T_r , is assumed to be 0°C , from Eq.(3.7.34),

$$T_m = \frac{\sum_{j=1}^n (V_j \cdot T_{i,j+1/2})}{V} \quad 3.7.35$$

where

$$T_{i,j+1/2} = \frac{T_{i,j+1} + T_{i,j}}{2}$$

In the case of a rectangular/slab geometry, $V_1 = V_2 = V_3 = \dots = V_n$ and

$$V = n \cdot V_j \quad 3.7.36$$

For a cylindrical geometry of unit length,

$$V_j = \pi \cdot (r_j^2 - r_{j+1}^2) = \pi \cdot \Delta r [2 \cdot R - (2j - 1) \cdot \Delta r] \quad 3.7.37$$

$$\text{and } V = \pi R^2 \quad 3.7.38$$

For a spherical geometry,

$$V_j = \frac{4}{3} \pi (r_j^3 - r_{j+1}^3) = \frac{4}{3} \pi \Delta r [\Delta r^2 + 3(R - (j-1) \Delta r)(R - j \Delta r)] \quad 3.7.39$$

$$V = \frac{4}{3} \pi R^3 \quad 3.7.40$$

Fully implicit method for solving Eq. (3.7.4)

This method is slightly more time consuming than the simple explicit method, but it compensates when the process is a slower one to reach steady state by permitting a much larger time step and thus saving a considerable computing time. Also, in the explicit method, $T_{i+1,j}$ depends only on $T_{i,j-1}$, $T_{i,j}$, and $T_{i,j+1}$ (Figure 3.7.1), where, in fact, other neighboring temperatures also do have some influence on $T_{i+1,j}$. This limitation is overcome by the fully implicit method, which thus ensures higher precision.

The finite difference representation of the left hand side of Eq. (3.7.7) remains unaltered as in the case of the explicit method. The right hand side of Eq. (3.7.11) is expressed in one advanced step in time. So, Eq. (3.7.11) in finite difference form can be rewritten as,

$$-\frac{\partial}{\partial r}(q \cdot A) = -\frac{[(q \cdot A)_{i+1,j+1/2} - (q \cdot A)_{i+1,j-1/2}]}{\Delta r} \quad 3.7.41$$

From Eq. (3.7.5), using the finite difference notations,

$$q_{i+1,j+1/2} = -k \cdot \frac{T_{i+1,j+1} - T_{i+1,j}}{\Delta r} \quad 3.7.42$$

and

$$q_{i+1,j-1/2} = -k \cdot \frac{T_{i+1,j} - T_{i+1,j-1}}{\Delta r} \quad 3.7.43$$

Now, Eq. (3.7.4) can be written using Eq. (3.7.41) with Eqs. (3.7.42) and (3.7.43) as the following,

$$\rho \cdot C_p \cdot A_j \cdot \frac{T_{i+1,j} - T_{i,j}}{\Delta t} = \left[k \cdot \frac{T_{i+1,j+1} - T_{i+1,j}}{\Delta r^2} \cdot A_{j+1/2} - k \cdot \frac{T_{i+1,j} - T_{i+1,j-1}}{\Delta r^2} \cdot A_{j-1/2} \right] \quad 3.7.44$$

Rearranging Eq. (3.7.44),

$$C_1 \cdot T_{i+1,j-1} + C_2 \cdot T_{i+1,j} + C_3 \cdot T_{i+1,j+1} = DD \cdot T_{i,j} \quad 3.7.45$$

where

$$\begin{aligned} DD &= -\frac{\Delta r^2}{\alpha \cdot \Delta t} \\ C_1 &= \frac{A_j + A_{j-1}}{2 \cdot A_j} \\ C_2 &= DD - (C_1 + C_3) \\ C_3 &= \frac{A_{j+1} + A_j}{2 \cdot A_j} \end{aligned}$$

Now, $T_{i,1}$ refers to surface temperature and $T_{i,n+1}$ refers to temperature at the other surface, which is the center point for a cylindrical or a spherical geometry, when exposed to a uniform atmosphere.

Equations at the boundaries

- (i) One side maintained at constant temperature, and the other side with derivative boundary condition.

Case 1. Side 1 is at T_{c1} and side 2 is with derivative BC:

Thus in Eq. (3.7.45) using boundary condition (3.7.21) in side 1,

for $j = 2$,

$$C_2 \cdot T_{i+1,2} + C_3 \cdot T_{i+1,3} = DD \cdot T_{i,2} - C_1 \cdot T_{c1} \quad 3.7.46$$

For $j = 3$,

$$C_1 \cdot T_{i+1,2} + C_2 \cdot T_{i+1,3} + C_3 \cdot T_{i+1,4} = DD \cdot T_{i,2}$$

For $j = 4$,

$$C_1 \cdot T_{i+1,2} + C_2 \cdot T_{i+1,3} + C_3 \cdot T_{i+1,4} = DD \cdot T_{i,3}$$

and for $j = n$,

$$C_1 \cdot T_{i+1,n-1} + C_2 \cdot T_{i+1,n} + C_3 \cdot T_{i+1,n+1} = DD \cdot T_{i,n},$$

where $T_{i+1,n+1}$ according to Eq. (3.7.23) is given by

$$T_{i+1,n+1} = T_{i+1,n} - \beta \cdot \Delta r$$

This makes the n -th equation as

$$C_1 \cdot T_{i+1,n-1} + (C_2 + C_3) \cdot T_{i+1,n} = DD \cdot T_{i,n} - C_3 \cdot \beta \cdot \Delta r \quad 3.7.47$$

In matrix form, this set of equations can be expressed as

$$\begin{bmatrix} C_2 & C_3 & & & \\ C_1 & C_2 & C_3 & & \\ & C_1 & C_2 & C_3 & \\ \dots & & & & \\ & & C_1 & C_2 & C_3 \\ & & C_1 & (C_2 + C_3) & \end{bmatrix} \begin{bmatrix} T_{i+1,2} \\ T_{i+1,3} \\ T_{i+1,4} \\ \dots \\ T_{i+1,n-1} \\ T_{i+1,n} \end{bmatrix} = \begin{bmatrix} DD \cdot T_{i,2} - C_1 \cdot T_{c1} \\ DD \cdot T_{i,3} \\ DD \cdot T_{i,4} \\ \dots \\ DD \cdot T_{i,n-1} \\ DD \cdot T_{i,n} + C_3 \cdot \beta \cdot \Delta r \end{bmatrix} \quad 3.7.48$$

Case 2. Side 1 is with derivative BC and side 2 is at T_{c2} :

In side 1, applying Eq. (3.7.22) at one advanced time step with Eq. (3.7.45) for $j = 2$,

$$(C_1 + C_2) \cdot T_{i+1,2} + C_3 \cdot T_{i+1,3} = DD \cdot T_{i,2} - C_1 \cdot \beta \cdot \Delta r \quad 3.7.49$$

and in side 2, for $j = n$ in Eq. (3.7.45) and applying $T_{i+1,n+1} = T_{c2}$

$$C_1 \cdot T_{i+1,n-1} + C_2 \cdot T_{i+1,n} = DD \cdot T_{i,n} - C_3 \cdot T_{c2} \quad 3.7.50$$

In matrix form,

$$\begin{bmatrix} (C_1 + C_2) & C_3 & & & \\ C_1 & C_2 & C_3 & & \\ & C_1 & C_2 & C_3 & \\ \dots & & & & \\ & & C_1 & C_2 & C_3 \\ & & C_1 & C_2 & \end{bmatrix} \begin{bmatrix} T_{i+1,2} \\ T_{i+1,3} \\ T_{i+1,4} \\ \dots \\ T_{i+1,n-1} \\ T_{i+1,n} \end{bmatrix} = \begin{bmatrix} DD \cdot T_{i,2} - C_1 \cdot \beta \cdot \Delta r \\ DD \cdot T_{i,3} \\ DD \cdot T_{i,4} \\ \dots \\ DD \cdot T_{i,n-1} \\ DD \cdot T_{i,n} + C_3 \cdot T_{c2} \end{bmatrix} \quad 3.7.51$$

Derivative boundary conditon with an imaginary node

As discussed earlier, the number of equations can be increased by one, thus letting the subscript, j go from **1** to **n** for derivative B.C. at side 1 or from **2** to **(n+1)** for derivative condition at side 2.

Case A. Side 1 is at derivative boundary conditon with imaginary node and side 2 at T_{c2} .

The imaginary node is the 0-th and Eq. (3.7.24) at one advanced time step is

$$T_{i+1,0} = T_{i+1,2} + 2\beta \cdot \Delta r$$

Substituting $j = 1$ in Eq. (3.7.45),

$$C_1 \cdot T_{i+1,0} + C_2 \cdot T_{i+1,1} + C_3 \cdot T_{i+1,2} = DD \cdot T_{i,1}$$

Now, applying the expression for $T_{i+1,0}$

$$C_2 \cdot T_{i+1,1} + (C_1 + C_3) \cdot T_{i+1,2} = DD \cdot T_{i,1} - 2C_1 \beta \cdot \Delta r \quad 3.7.52$$

and the equation at side 2 is identical to Eq. (3.7.50).

Case B. Side 2 is at derivative boundary condition with imaginary node and side 1 at T_{c1} .

The imaginary node is the $(n+2)$ -th and Eq. (3.7.25) at one advanced time step is

$$T_{i+1,n+2} = T_{i+1,n} - 2\beta \cdot \Delta r$$

Substituting $j = n + 1$ in Eq. (3.7.45),

$$C_1 \cdot T_{i+1,n} + C_2 \cdot T_{i+1,n+1} + C_3 \cdot T_{i+1,n+2} = DD \cdot T_{i,n+1}$$

Now, applying the expression for $T_{i+1,n+2}$

$$C_2 \cdot T_{i+1,n+1} + (C_1 + C_3) \cdot T_{i+1,n} = DD \cdot T_{i,n+1} + 2C_3 \beta \cdot \Delta r \quad 3.7.53$$

and the equation at side 1 is identical to Eq. (3.7.46).

In matrix form, the set of equations for Case A is represented as follows:

$$\begin{bmatrix} C_2 & (C_1 + C_3) \\ C_1 & C_2 & C_3 \\ & C_1 & C_2 & C_3 \\ \dots & \dots & \dots & \dots \\ & C_1 & C_2 & C_3 \\ & & C_1 & C_2 \end{bmatrix} \begin{bmatrix} T_{i+1,2} \\ T_{i+1,3} \\ T_{i+1,4} \\ \dots \\ T_{i+1,n-1} \\ T_{i+1,n} \end{bmatrix} = \begin{bmatrix} DD \cdot T_{i,1} - 2C_1 \beta \cdot \Delta r \\ DD \cdot T_{i,2} \\ DD \cdot T_{i,3} \\ \dots \\ DD \cdot T_{i,n-1} \\ DD \cdot T_{i,n} - C_3 \cdot T_{c2} \end{bmatrix} \quad 3.7.54$$

and for Case B, in matrix form, the set of equations is represented as follows:

$$\begin{bmatrix} C_2 & C_3 & & & \\ C_1 & C_2 & C_3 & & \\ & C_1 & C_2 & C_3 & \\ \dots & & & & \\ & & C_1 & C_2 & C_3 \\ & & (C_1 + C_3) & C_2 & \end{bmatrix} \begin{bmatrix} T_{i+1,2} \\ T_{i+1,3} \\ T_{i+1,4} \\ \dots \\ T_{i+1,n-1} \\ T_{i+1,n} \end{bmatrix} = \begin{bmatrix} DD \cdot T_{i,2} - C_1 \cdot T_{c1} \\ DD \cdot T_{i,3} \\ DD \cdot T_{i,4} \\ \dots \\ DD \cdot T_{i,n} \\ DD \cdot T_{i,n+1} + 2C_3 \cdot \beta \cdot \Delta r \end{bmatrix} \quad 3.7.55$$

When closely examined, it is observed that the equations linked with imaginary node(s), need the values of the coefficients C_1 , C_2 , and C_3 in Eq. (3.7.45) evaluated at $j = n+1$. Now, these coefficients have the area value at the j -th node as their denominator, which should not be allowed to be zero. Unfortunately, this is the case at the center of a cylinder or a sphere, hence the imaginary node concept cannot be applied in such cases. To maintain a uniform treatment, this concept has not been used in the individual programs developed for each of the three methods, namely, explicit, fully implicit, and Crank-Nicolson listed in Appendices A.3.7.1 through A.3.7.3. The reader, however, may try to use the imaginary node concept where possible to get an idea of the difference that it makes with the approach used here. This approach will be used later in the general program developed.

In the Crank-Nicolson method, which will be discussed later in this chapter, the value of $T_{i,j+1}$ is required to be evaluated at each time step. For the imaginary $(n+2)$ -th node, that is, for $j = n+1$, the value of $T_{i,n+2}$ should be available for this method to work. This is done by using Eq.(3.7.25) for this fictitious temperature. The program **ALTDIR.FOR** listed in Appendix A.3.7.4, however, uses this concept where applicable with the exception of derivative BC at the center of a solid cylindrical or a spherical geometry.

- (ii) One side maintained at constant temperature, T_{c1} and the other side at T_{c2} .

In this case, for $j = 2$, Eq. (3.7.46) and for $j = n$, Eq. (3.7.50) apply. In matrix form, this set of equations can be expressed as

$$\begin{bmatrix} C_2 & C_3 & & & \\ C_1 & C_2 & C_3 & & \\ & C_1 & C_2 & C_3 & \\ \dots & & & & \\ & & C_1 & C_2 & C_3 \\ & & C_1 & C_2 & \end{bmatrix} \begin{bmatrix} T_{i+1,2} \\ T_{i+1,3} \\ T_{i+1,4} \\ \dots \\ T_{i+1,n-1} \\ T_{i+1,n} \end{bmatrix} = \begin{bmatrix} DD \cdot T_{i,2} - C_1 \cdot T_{c1} \\ DD \cdot T_{i,3} \\ DD \cdot T_{i,4} \\ \dots \\ DD \cdot T_{i,n-1} \\ DD \cdot T_{i,n} - C_3 \cdot T_{c2} \end{bmatrix} \quad 3.7.56$$

When $T_{c1} = T_{c2}$, the set of equations is represented by matrix (3.7.48) and the half the geometry should be solved.

(iii) One side maintained at a constant temperature and the other side exposed to free convection.

Case 1. Side 1 is at T_{c1} and side 2 with convective BC:

The equation at side 1 is the same as Eq. (3.7.46).

Now, at side 2, for $j = n$

$$C_1 \cdot T_{i+1,n-1} + C_2 \cdot T_{i+1,n} + C_3 \cdot T_{i+1,n+1} = DD \cdot T_{i,n}$$

where $T_{i+1,n+1}$ is given by Eq. (3.7.30), at one advanced time step, which is rewritten as

$$T_{i+1,n+1} = \frac{T_{c2} + DD2 \cdot T_{i+1,n}}{1 + DD2} \quad 3.7.57$$

where DD2 is given by Eq. (3.7.31).

Now, Eq. (3.7.57) after multiplying by C_3 , can be expressed as

$$C_3 \cdot T_{i+1,n+1} = CC_1 + CC_2 \cdot T_{i+1,n}$$

where

$$CC_1 = \frac{T_{c2} \cdot C_3}{1 + DD2} ; \text{ and } CC_2 = \frac{DD2 \cdot C_3}{1 + DD2}$$

So, the n -th equation becomes

$$C_1 \cdot T_{i+1,n-1} + (C_2 + CC_2) \cdot T_{i+1,n} = DD \cdot T_{i,n} - CC_1 \quad 3.7.58$$

In matrix form, this set of equations can be expressed as

$$\begin{bmatrix} C_2 & C_3 & & & \\ C_1 & C_2 & C_3 & & \\ & C_1 & C_2 & C_3 & \\ \dots & \dots & \dots & \dots & \dots \\ & & C_1 & C_2 & C_3 \\ & & C_1 & (C_2 + CC_2) & \end{bmatrix} \begin{bmatrix} T_{i+1,2} \\ T_{i+1,3} \\ T_{i+1,4} \\ \dots \\ T_{i+1,n-1} \\ T_{i+1,n} \end{bmatrix} = \begin{bmatrix} DD \cdot T_{i,2} - C_1 \cdot T_{c1} \\ DD \cdot T_{i,3} \\ DD \cdot T_{i,4} \\ \dots \\ DD \cdot T_{i,n-1} \\ DD \cdot T_{i,n} - CC_1 \end{bmatrix} \quad 3.7.59$$

Case 2. Side 2 is at T_{c2} and side 1 with convective BC:

The equation at side 2 is the same as Eq. (3.7.50).

Now, at side 1, for $j = 2$

$$C_1 \cdot T_{i+1,1} + C_2 \cdot T_{i+1,2} + C_3 \cdot T_{i+1,3} = DD \cdot T_{i,2}$$

where $T_{i+1,1}$ is given by Eq. (3.7.32), at one advanced time step, which is rewritten as

$$T_{i+1,1} = \frac{T_{c1} + DD1 \cdot T_{i+1,2}}{1 + DD1} \quad 3.7.60$$

where $DD1$ is given by Eq. (3.7.33).

Now, Eq. (3.7.60), after multiplying by C_1 , can be expressed as

$$C_1 \cdot T_{i+1,1} = CC + CC_0 \cdot T_{i+1,2}$$

where

$$CC = \frac{T_{c1} \cdot C_1}{1 + DD1} ; \quad \text{and} \quad CC_0 = \frac{DD1 \cdot C_1}{1 + DD1}$$

So, the first equation becomes

$$(C_2 + CC_0) \cdot T_{i+1,2} + C_3 \cdot T_{i+1,3} = DD \cdot T_{i,2} - CC \quad 3.7.61$$

In matrix form this set of equations can be expressed as

$$\begin{bmatrix} (C_2 + CC_0) & C_3 & & & \\ C_1 & C_2 & C_3 & & \\ & C_1 & C_2 & C_3 & \\ \dots & & & & \\ & & C_1 & C_2 & C_3 \\ & & & C_1 & C_2 \end{bmatrix} \begin{bmatrix} T_{i+1,2} \\ T_{i+1,3} \\ T_{i+1,4} \\ \dots \\ T_{i+1,n-1} \\ T_{i+1,n} \end{bmatrix} = \begin{bmatrix} DD \cdot T_{i,2} - CC \\ DD \cdot T_{i,3} \\ DD \cdot T_{i,4} \\ \dots \\ DD \cdot T_{i,n-1} \\ DD \cdot T_{i,n} - C_3 \cdot T_{c2} \end{bmatrix} \quad 3.7.62$$

(iv) Both sides are exposed to free convection.

For convection from side 1 and 2, Eqs. (3.7.61) and (3.7.58), respectively, are directly applicable. In matrix form, this set of equations can be expressed as

$$\begin{bmatrix}
 (C_2 + CC_0) & C_3 & & & \\
 C_1 & C_2 & C_3 & & \\
 & C_1 & C_2 & C_3 & \\
 \dots & & & & \\
 & & C_1 & C_2 & C_3 \\
 & & C_1 & (C_2 + CC_2) &
 \end{bmatrix}
 \begin{bmatrix}
 T_{i+1,2} \\
 T_{i+1,3} \\
 T_{i+1,4} \\
 \dots \\
 T_{i+1,n-1} \\
 T_{i+1,n}
 \end{bmatrix}
 =
 \begin{bmatrix}
 DD.T_{i,2} - CC \\
 DD.T_{i,3} \\
 DD.T_{i,4} \\
 \dots \\
 DD.T_{i,n-1} \\
 DD.T_{i,n} - CC_1
 \end{bmatrix}
 \quad 3.7.63$$

When for both $T_{c1} = T_{c2}$ and $h_1 = h_2$, due to symmetry, only half of the pertinent dimension is solved, and we get convective boundary condition in side 1 where Eq. (3.7.61) directly applies.

For cylindrical and spherical geometries exposed to uniform outside temperature, the above mentioned two boundary conditions hold good and the matrix representation of the set of equations to be solved is given by :

$$\begin{bmatrix}
 (C_2 + CC_0) & C_3 & & & \\
 C_1 & C_2 & C_3 & & \\
 & C_1 & C_2 & C_3 & \\
 \dots & & & & \\
 & & C_1 & C_2 & C_3 \\
 & & C_1 & (C_2 + C_3) &
 \end{bmatrix}
 \begin{bmatrix}
 T_{i+1,2} \\
 T_{i+1,3} \\
 T_{i+1,4} \\
 \dots \\
 T_{i+1,n-1} \\
 T_{i+1,n}
 \end{bmatrix}
 =
 \begin{bmatrix}
 DD.T_{i,2} - CC \\
 DD.T_{i,3} \\
 DD.T_{i,4} \\
 \dots \\
 DD.T_{i,n-1} \\
 DD.T_{i,n}
 \end{bmatrix}
 \quad 3.7.64$$

All the matrices so far developed for the fully implicit method (Eqs. 3.7.48, 3.7.5, 3.7.54, 3.7.55, 3.7.56, 3.7.59, 3.7.62, 3.7.63, and 3.7.64) are tridiagonal. This type of banded matrix can easily be solved by the Gauss elimination method described in Chapter 3.4.

Crank-Nicolson method

As was explained earlier, the fully implicit method, at the cost of slightly extra calculations, is more stable and precise than the straight-forward explicit method. Since both these methods use same-order finite difference approximations for the derivatives, they suffer the same order of discretization error in this respect, that is, time dependency is of the first-order in Δt , whereas space dependency is of the second-order in Δr . The Crank-Nicolson scheme, on the other hand, reduces the time dependency from first-order to second-order, whereas the space dependency does not alter.

Precisely what is done in the Crank-Nicolson method is that the left hand side derivative of Eq. (3.7.41) is expressed by the finite difference approximation at the present time step, just as in the explicit method, as well

as at one advanced step in time as in the fully implicit method, and, finally, an arithmetic average is made of the two. The first derivative of temperature with respect to time, on the other hand, is expressed the same way it was done in the case of the explicit or the fully implicit scheme. Now,

$$-\frac{\partial}{\partial t}(q \cdot A) = -\frac{[(q \cdot A)_{i+1,j+1/2} - (q \cdot A)_{i+1,j-1/2}]}{\Delta r}$$

where

$$q_{i+1,j+1/2} = -\frac{1}{2}k \left[\frac{T_{i+1,j+1} - T_{i+1,j}}{\Delta r} + \frac{T_{i,j+1} - T_{i,j}}{\Delta r} \right]$$

and

$$q_{i+1,j-1/2} = -\frac{1}{2}k \left[\frac{T_{i+1,j} - T_{i+1,j-1}}{\Delta r} + \frac{T_{i,j} - T_{i,j-1}}{\Delta r} \right]$$

$$\begin{aligned} \text{So, } \rho \cdot C_p \cdot A_j \cdot \frac{T_{i+1,j} - T_{i,j}}{\Delta t} &= \frac{1}{2}k \cdot A_{j+1/2} \left[\frac{T_{i+1,j+1} - T_{i+1,j}}{\Delta r^2} + \frac{T_{i,j+1} - T_{i,j}}{\Delta r^2} \right] - \\ &\quad \frac{1}{2}k \cdot A_{j-1/2} \left[\frac{T_{i+1,j} - T_{i+1,j-1}}{\Delta r^2} + \frac{T_{i,j} - T_{i,j-1}}{\Delta r^2} \right] \end{aligned} \quad 3.7.65$$

After a number of mathematical manipulations, the final expression of the Crank-Nicolson scheme is obtained as

$$C_1 \cdot T_{i+1,j-1} + C_2 \cdot T_{i+1,j} + C_3 \cdot T_{i+1,j+1} = DDD \cdot T_{i,j} - C_1 \cdot T_{i,j-1} - C_3 \cdot T_{i,j+1} \quad 3.7.66$$

where

$$\begin{aligned} C_2 &= 2 \cdot DD - C_1 - C_3 \\ DDD &= 2 \cdot DD + C_1 + C_3 \end{aligned}$$

and C_1 , C_3 , and DD are the same as defined earlier in relation to Eq. (3.7.45).

The Crank-Nicolson representation is also a form of the fully implicit method, and Eq. (3.7.66) is very similar to Eq. (3.7.45) for the fully implicit scheme. The only difference lies in the right hand sides of the equations, which can be easily taken care of. This method is not as totally unconditionally stable as the fully implicit method.

Development of a general expression

The weight on each the two finite difference approximations of the same derivative on the right hand side of Eq. (3.7.65) is unity, and thus the average is made by dividing their sum by 2. To make it a general one which can be

easily transformed into any of the expressions so far discussed, e.g., explicit, fully implicit or Crank-Nicolson, the first finite difference approximation is multiplied by a number **m** and the second by **(1-m)** so that their sum is divided by $(m + 1 - m)$, that is, unity. This **m** and **(1-m)** are called **weights** and **m** is also termed the **degree of implicitness**. Now, substituting these weights into Eq. (3.7.65), the following expression is obtained:

$$\rho \cdot C_p \cdot A_j \cdot \frac{T_{i+1,j} - T_{i,j}}{\Delta t} = k \cdot A_{j+1/2} \left[m \cdot \frac{T_{i+1,j+1} - T_{i+1,j}}{\Delta r^2} + (1-m) \cdot \frac{T_{i,j+1} - T_{i,j}}{\Delta r^2} \right] - k \cdot A_{j-1/2} \left[m \cdot \frac{T_{i+1,j} - T_{i+1,j-1}}{\Delta r^2} + (1-m) \cdot \frac{T_{i,j} - T_{i,j-1}}{\Delta r^2} \right]$$

After rearrangement, the following general expression is obtained:

$$C_{11} \cdot T_{i+1,j-1} + C_{22} \cdot T_{i+1,j} + C_{33} \cdot T_{i+1,j+1} = DDD \cdot T_{i,j} - (1-m) \cdot (C_1 \cdot T_{i,j-1} + C_3 \cdot T_{i,j+1}) \quad 3.7.67$$

where

$$\begin{aligned} C_{11} &= m \cdot C_1 \\ C_{22} &= DD - m \cdot (C_1 + C_3) \\ C_{33} &= m \cdot C_3 \\ DDD &= DD + (1-m) \cdot (C_1 + C_3) \end{aligned}$$

and C_1 , C_3 and DD are same as defined earlier.

Now, for $m = 0$, Eq. (3.7.67) becomes an explicit expression,
for $m = 1$, Eq. (3.7.67) becomes a fully implicit expression,
and for $m = 0.5$, Eq. (3.5.67) becomes the Crank-Nicolson's expression.

The value of the degree of implicitness, **m**, can be varied other than the ones mentioned above and accordingly expressions are obtained of various discretization error bounds on Δr .

Development of computer programs

Separate programs were developed in FORTRAN for the three methods discussed. The subroutine **GEOMTY(L)** listed in Appendix A.3.7.2 calculates the area and volume of different configurations. It is common for the main programs dealing with fully implicit and Crank-Nicolson schemes. The source code listings of the explicit (**EXPLCT.FOR**), fully implicit (**IMPLCT.FOR**), and Crank-Nicolson (**CRANK.FOR**) methods are given in Appendices A.3.7.1, A.3.7.4, and A.3.7.5, respectively. The main program

parts of these three are almost identical. In all the programs, a general expression for any regular geometry has been used and side 1 (external surface for cylindrical and spherical geometry) has been maintained at a constant temperature, **TC** and the other side is insulated. Obviously, for cylindrical and spherical geometry, side 2 is the center and derivative condition prevails there with temperature gradient, **BETA** as zero. For simplicity, this gradient has been taken as zero for the slab geometry as well, that is, perfect insulation is considered. The subroutine **EXPLCT** uses Eq. (3.7.16), whereas subroutines **COEFF** and **TRIDGL** use Eq. (3.7.45) for the fully implicit method and Eq. (3.7.66) for the Crank-Nicolson method. The subroutine **TRIDGL** was developed earlier in Chapter 3.4 and is again listed in Appendix A.3.7.3 as **TRIDGL.FOR**.

The **Diagonal**, **Below-diagonal**, and **Above-diagonal** elements of banded coefficient matrix (tridiagonal matrix) have been represented in the program by **D(j)**, **B(j)**, and **A(j)**, respectively where **j** goes from **2** to **n**. The elements of column vectors on the right hand side of all the matrices (Eqs. 3.7.48, 3.7.5, 3.7.54, 3.7.55, 3.7.56, 3.7.59, 3.7.62, 3.7.63, and 3.7.64) have been represented by **R(j)**. All these coefficients are calculated by the subroutine **COEFF**.

Thermophysical properties (density-**RO**, specific heat-**CP** and thermal conductivity-**XK**) are provided by a separate FORTRAN file named **THERMO.FOR**. It should be created by the user before compiling, and an example listing is given in Appendix A.3.7.8. A careful consistency in all the units used is advised. The dimension of the dependent variable **T**, has been limited to **2** for the time step and **101** for the step in space. Thus, the maximum node points allowed has been limited to 100. The number of steps in time depends on the time step and total time of exposure, and it may be too large at times. To avoid a large time-dimension on **T**, after each time step, calculated values of **T** are rearranged as their values at one step earlier in time. Double precision arithmetic has been used to reduce roundoff error especially in implicit and Crank-Nicolson methods.

Initial and boundary conditions are provided by an external function, **FCTN.FOR**, which should be created by the user before compiling any of the programs in this chapter. An example listing of this function is included in Appendix A.3.7.9. This provides the advantage of using an initial condition as a function of position, and time dependent boundary conditions. The function **BC** has two arguments — **I** and **TX**, the first one being the identifier and the second one the variable. The initial condition is identified with **I = 0**. If it is not a function of position, or, in other words, if it is constant throughout the geometry, the same function is used by ascribing itself that constant value. The variable **TX** should not appear in the function statement in that case. Otherwise **TX** can be used for both position (initial condition, **I = 0**) and time (boundary condition, **I > 0**).

All these three programs, after being compiled and run, ask the user about the option on the product geometry, namely rectangular, cylindrical or

spherical. Once chosen, they ask about initial temperature and temperature at one side, and thereafter the principal dimension, total time of exposure, and number of divisions along the principal dimension. In any of the three methods, the user has to provide the value of the time step. The time step is calculated by Eq. (3.7.10) for the explicit method, but the user is still asked to give its value less than or equal to the time step calculated for stability. Finally, they offer an option to see step-by-step results or the final results. If the step-by-step results are chosen, printing interval, that is, the number of time steps to be skipped, is asked to be given. To facilitate this decision, the user is shown the time step chosen or calculated and the maximum number of time steps that can be skipped.

The results show the name of the scheme used, the geometry of the product, and the boundary condition(s) followed by the mean temperature calculated by Eq. (3.7.35) and the time of exposure. Finally nodal temperature values are listed. All this information is also stored in a file named **DATA.OUT** created by the program itself during compilation. The following problem is solved by each of the programs developed:

Example 3.7.1. A nondimensional slab geometry (thickness = 1, diffusivity unity, and minimum and maximum nondimensional temperature as zero and unity, respectively) and both sides of which are maintained at $T = 0$ (Dirichlet condition). Its initial nondimensional temperature is 1. Calculate the temperature at the center of the slab at different times.

Solution:

Programs **EXPLCT.FOR**, **IMPLCT.FOR** and **CRANK.FOR** are used to solve the problem using different grid spacing and time steps. Two grid spacing — one with 10 and the other with 20 divisions for the entire width and different time steps (0.005 and 0.001 for explicit and 0.5, 0.01 and 0.001 for the other two implicit schemes) are used. Only half of the slab geometry is to be solved because the problem is symmetric with respect to thickness.

A noticeable difference is observed among the values calculated by the explicit and the two versions of the implicit methods as expected. The time step, in the case of implicit methods, is gradually increased to check its effect on the accuracy in the results obtained. Table 3.7.1 gives a comparison among the three methods when the time step is varied.

The fully implicit method is seen to be stable and more accurate than the Crank-Nicolson up to an optimum time step size. The Explicit and Crank-Nicolson methods are not unconditionally stable. When the step size is further reduced, the Crank-Nicolson gets better with respect to accuracy. In general, as the time step is reduced or the number of grid points is increased, the accuracy gets better. Estimation gets worse as the solution proceeds along time. In fact, the use of the backward finite difference representation for the first derivative in the derivative boundary condition given by Eq. (3.7.23) is

Table 3.7.1. A comparison among the three methods of solving Problem 3.7.1.

Method	Value of dependent variable at the center									
	t = 0.001		t = 0.01		t = 0.1		t = 0.2		t = 1.0	
	n = 5	n = 10	n = 5	n = 10	n = 5	n = 10	n = 5	n = 10	n = 5	n = 10
<u>Explicit</u>										
a. $\Delta t=0.005$	—	—	1.00000	*	0.35758	*	0.10310	*	0.00000	*
b. $\Delta t=0.001$	1.00000	1.00000	0.99098	0.99916	0.36881	0.42263	0.10960	0.14109	0.00001	0.00002
<u>Implicit</u>										
a. $\Delta t=0.5$	—	—	—	—	—	—	—	—	0.02499	0.03022
b. $\Delta t=0.005$	—	—	0.97757	0.98796	0.38465	0.43742	0.11927	0.15124	0.00001	0.00003
c. $\Delta t=0.001$	0.99995	1.00000	0.98560	0.99501	0.37420	0.42766	0.11283	0.14449	0.00001	0.00002
<u>Crank-Nicolson</u>										
a. $\Delta t=0.5$	—	—	—	—	—	—	—	—	*	*
b. $\Delta t=0.005$	—	—	0.98809	0.99603	0.37138	0.42505	0.11114	0.14272	0.00001	0.00002
c. $\Delta t=0.001$	0.99999	1.00000	0.98815	0.99701	0.37151	0.42515	0.11121	0.14279	0.00001	0.00002
<u>Analytical</u>	1.00000		0.99919		0.47449		0.17687		0.00007	

* scheme unstable.

not quite precise. This causes deviation in the center temperature from the true value and gradually error propagates in the entire profile. This error can be reduced by increasing the number of divisions along the principal dimension.

Development of a general program

The programs developed earlier are limited to handling only one type of boundary condition, namely Cauchy conditions, and only one method at a time. Now, a program is developed that uses the general expression given by Eq. (3.7.67) so that the subroutine **COEFF** in combination with **TRIDGL** works as an explicit scheme when the degree of implicitness, **XM** is zero, as an implicit scheme when **XM** is unity and as Crank-Nicolson scheme when **XM** is one half. This can take care of all the boundary conditions discussed so far, and, at the same time, the imaginary-node concept is utilized in the case of the slab geometry. In one case (one side is maintained at constant temperature and the other side is with Neumann conditions), the first derivative (gradient **BETA**) can be given a constant value, which includes zero.

The main program, listed in Appendix A.3.7.6 as **PDEQN.FOR**, also uses the same two external programs, namely, **THERMO.FOR** and **FCTN.FOR** listed in Appendices A.3.7.8 and A.3.7.9, respectively. Once it is compiled and executed, it offers the main menu options for the users to choose the finite difference method among (1) explicit, (2) fully implicit, and (3) Crank-Nicolson. The secondary menu thereafter asks to choose for product configuration, which includes (1) rectangular/slab, (2) cylindrical, and (3) spherical geometry. The subroutine **COEFF** calculates coefficients of the matrices that are used by **TRIDGL.FOR** to solve the set of equations.

The third menu helps decide the choice of boundary conditions at side 1 and 2, which include for slab geometry — (i) side 1 at constant temperature and side 2 with derivative BC, (ii) side 1 and 2 are at constant temperatures (these temperatures may be different or these may be equal; when equal, half of the geometry is solved due to symmetry), (iii) one side at constant temperature and the other side having convective BC, and (iv) both sides are exposed to free convection. When temperature and heat transfer coefficients on both sides are identical, the problem is solved automatically considering only half of the principal dimension due to symmetry. For the cylindrical and spherical geometries, it is assumed that these are exposed to one single outside atmosphere, thus attributing convective BC on their outside surface and obviously derivative BC at their center with temperature gradient, **BETA** as zero. The subroutine **BOUND(NN)** is used to supply boundary conditions to the main program. The counter **NN** is the number of divisions, which is equal to **N+1** when an imaginary node point is considered for the derivative (Neumann) boundary condition. The product temperature, temperature at the boundaries, heat transfer coefficient(s), temperature gradient, etc., are to be given when asked by the program depending on the specific case under consideration.

Once the product dimension (thickness in the case of rectangular geometry and radius for the others), total time of exposure, and number of divisions along the principal dimension are decided, it asks for the time step showing its calculated value (even in the case of the explicit method). The results can be printed step-by-step along time, or only final results. If step-by-step results are chosen, the user is asked to give the printing interval (number of time steps to be skipped) to save space. It also advises the user at this moment about the time step chosen and the maximum number of time steps that can be skipped.

The results include the same information that the previous programs supply. All these information are stored in a file named **DATA.OUT** created by the program. In Table 3.7.2, results are presented considering the imaginary node point (INP) concept.

It is noted from Table 3.7.2 that the INP concept brought the estimation closer to analytical results. In fact, the results for half of the geometry, using this concept, are identical to those obtained when the entire geometry is

considered, whereas it is not the case when the INP concept was not used (see Table 3.7.1). Thus this concept should be used where applicable.

Table 3.7.2. Solution of problem 3.7.1 with the INP concept.

Method	Value of dependent variable at the center									
	t = 0.001		t = 0.01		t = 0.1		t = 0.2		t = 1.0	
	n=5	n=10	n=5	n=10	n=5	n=10	n=5	n=10	n=5	n=10
<u>Explicit</u>										
a. $\Delta t=0.005$	-	-	1.00000	*	0.47445	*	0.17391	*	0.00006	*
b. $\Delta t=0.001$	1.00000	1.00000	0.99788	0.99979	0.47211	0.47217	0.17656	0.17550	0.00007	0.00006
<u>Implicit</u>										
a. $\Delta t=0.5$	-	-	-	-	-	-	-	-	0.03614	0.03600
b. $\Delta t=0.005$	-	-	0.98926	0.99154	0.48520	0.48551	0.18674	0.18574	0.00009	0.00008
c. $\Delta t=0.001$	0.99999	1.00000	0.99481	0.99709	0.47658	0.47673	0.17998	0.17894	0.00007	0.00007
<u>Crank-Nicolson</u>										
a. $\Delta t=0.5$	-	-	-	-	-	-	-	-	*	*
b. $\Delta t=0.005$	-	-	0.99579	0.99769	0.47428	0.47438	0.17820	0.17715	0.00007	0.00007
c. $\Delta t=0.001$	1.00000	1.00000	0.99632	0.99848	0.47435	0.47446	0.17827	0.17722	0.00007	0.00007
<u>Analytical</u>	1.00000		0.99919		0.47449		0.17687		0.00007	

* scheme unstable.

Alternating approach for solving cylindrical and spherical geometries

The approach utilized so far to solve these geometries has come from the general derivation of PDE, where, if desired, density, specific heat, and thermal conductivity (thus thermal diffusivity) can also be treated as functions of position and/or time if a more general expression is developed. The other rather limited approach is to use the change of variable concept to transform PDEs for cylindrical and spherical geometries to that for a slab geometry. It can be done as follows:

Cylindrical geometry

The governing PDE in this case is given by

$$\frac{\partial T}{\partial t} = \alpha \cdot \left(\frac{\partial^2 T}{\partial r^2} + \frac{1}{r} \frac{\partial T}{\partial r} \right) \quad 3.7.68$$

Now, defining another variable $R = \ln(r)$, Eq. (3.7.68) is transformed to

$$\frac{\partial T}{\partial t} = \frac{\alpha}{e^{2R}} \cdot \frac{\partial^2 T}{\partial R^2} \quad 3.7.69$$

Spherical geometry

The governing PDE in this case is given by

$$\frac{\partial T}{\partial t} = \alpha \cdot \left(\frac{\partial^2 T}{\partial r^2} + \frac{2}{r} \frac{\partial T}{\partial r} \right) \quad 3.7.70$$

Now, defining variable $T = W/r$, Eq. (3.7.70) is transformed to

$$\frac{\partial W}{\partial t} = \alpha \cdot \frac{\partial^2 W}{\partial r^2} \quad 3.7.71$$

Both Eqs. (3.7.69) and (3.7.71) suffer one more limitation than the one mentioned earlier; they exclude $r = 0$ — the center. Thus symmetrical heat flow problems concerning a hollow cylinder and sphere can be solved by expressing these equations in terms of the finite difference representations as already discussed.

The accuracy of the approach used here to deal with cylindrical and spherical geometry is checked using the analytical solution of a spherical object of radius 0.1 m initially at 25°C that is exposed to an environment so that its surface is maintained at 0°C. The local temperature at a distance 0.01 m from its center is calculated from the following analytical solution:

$$T = T_c + \frac{2 \cdot (T_o - T_c) \cdot R}{\pi \cdot r} \cdot \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} \cdot \sin(n \cdot \pi \cdot r / R) \cdot e^{-\alpha \cdot n^2 \cdot \pi^2 \cdot t / R^2}$$

where

R : radius of the solid sphere = 0.1 m

r : variable radius (radial position of interest) = 0.01 m

T_c : surface temperature = 0.0 °C

T_o : initial temperature = 25.0 °C

α : thermal diffusivity = 10^{-7} m²/s

t : time of exposure = 7200.0 s.

With the above data, the analytical solution gives $T = 21.51010^\circ\text{C}$, whereas the computer program **IMPLCT.FOR** gives it as 20.79936°C (relative error = 3.3042%) with a step size of 600 s and 10 divisions along

half of its radius. When the number of divisions was increased to 100, the numerical answer becomes 21.24870°C (relative error = 1.2152%). The error is further minimized when the step size is also reduced to 10 s keeping the number of divisions as 100 (an estimated value of 21.49986°C with a relative error of 0.0476%). This establishes the earlier statement that increasing of number of divisions improves accuracy of the approach without the imaginary-node concept, which is the case here.

The average temperature calculated by Eq. (3.7.35) is found to be very close to that obtained by the following analytical expression:

$$T_{av} = T_c + (T_o - T_c) \cdot \frac{6}{\pi^2} \cdot \sum_{n=1}^{\infty} \frac{1}{n^2} \cdot e^{-\alpha \cdot n^2 \cdot \pi^2 \cdot t / R^2}$$

After 2 hours (7200 s) of exposure, T_{av} is calculated by the above equation as 7.691805°C, and numerically by **IMPLCT.FOR**, it is obtained as 7.69600 °C (a relative error of -0.0545%) with a step size of 10 s and 100 divisions along the radius of the sphere.

Solution of a two-dimensional parabolic partial differential equation

In Cartesian coordinates, a two-dimensional heat conduction equation is written as

$$\frac{\partial T}{\partial t} = \alpha \cdot \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) \quad 3.7.72$$

It needs one initial and four boundary conditions (two in the x-direction and two in the y-direction) to solve such equations.

Now, using the subscript **i** for step along the x-direction, **j** in the y-direction and **k** for the time dimension, the first term on the right hand side of Eq. (3.7.72) is expressed by a central difference representation, as usual keeping **j** at the present step in **x**, whereas the second term is expressed keeping **i** at the present step in **y**. The left hand side is expressed the same way it was in one dimension. Thus

$$\frac{T_{i,j,k+1} - T_{i,j,k}}{\Delta t} = \alpha \cdot \left(\frac{T_{i+1,j,k} - 2 \cdot T_{i,j,k} + T_{i-1,j,k}}{\Delta x^2} + \frac{T_{i,j+1,k} - 2 \cdot T_{i,j,k} + T_{i,j-1,k}}{\Delta y^2} \right)$$

After rearranging and letting $\Delta x = \Delta y$ for a square grid,

$$T_{i,j,k+1} = D \cdot T_{i-1,j,k} + (1 - 4 \cdot D) \cdot T_{i,j,k} + D \cdot T_{i+1,j,k} + D \cdot T_{i,j-1,k} + D \cdot T_{i,j+1,k} \quad 3.7.73$$

where

$$D = \frac{\alpha \cdot \Delta t}{\Delta x^2}$$

The usual explicit scheme can be used to solve Eq. (3.7.73) with the following restriction on the time step for stability:

$$\Delta t \leq \frac{0.5}{\alpha [\Delta x^{-2} + \Delta y^{-2}]} \quad 3.7.74$$

for a square grid,

$$\Delta t \leq \frac{0.25 \cdot \Delta x^2}{\alpha}$$

The implicit representation, on the other hand, is given by

$$\frac{T_{i,j,k+1} - T_{i,j,k}}{\Delta t} = \alpha \cdot \left(\frac{T_{i+1,j,k+1} - 2 \cdot T_{i,j,k+1} + T_{i-1,j,k+1}}{\Delta x^2} + \frac{T_{i,j+1,k+1} - 2 \cdot T_{i,j,k+1} + T_{i,j-1,k+1}}{\Delta y^2} \right) \quad 3.7.75$$

A usual practice is to take the step size along the x-direction equal to that in the y-direction, that is, $\Delta x = \Delta y$. Substituting this into Eq. (3.7.75) and substituting $(\alpha \cdot \Delta t / \Delta x^2)$ as D ,

$$-D \cdot T_{i-1,j,k+1} - D \cdot T_{i,j-1,k+1} + (1 + 4 \cdot D) \cdot T_{i,j,k+1} - D \cdot T_{i+1,j,k+1} - D \cdot T_{i,j+1,k+1} = T_{i,j,k} \quad 3.7.76$$

Unlike the one-dimensional representation, Eq. (3.7.76) has five unknowns and the usual tridiagonal system does not work as before. However, the set of such equations can be solved by the usual Gauss elimination method with a full matrix, which will require a considerable computer time or by the Gauss-Seidel iterative method discussed in Chapter 3.4. It has been observed that the Gauss-Seidel method requires a considerable number of iterations to converge in solving a set of such five-element equations.

The **Alternating-Direction Implicit (ADI)** method removes all these disadvantages and, at the same time, allows the use of a tridiagonal coefficient matrix to yield a straightforward solution. In this method, two

finite difference representations of Eq. (3.7.72) are made, each at a half time step. The first representation is implicit in the x-direction whereas the second is implicit in the y-direction. If $T_{i,j,k+1/2}$ is an intermediate value between the k-th and (k+1)-th time steps, the first equation implicit in only the x-direction can be expressed as

$$\frac{T_{i,j,k+1/2} - T_{i,j,k}}{\Delta t / 2} = \alpha \cdot \left(\frac{T_{i+1,j,k+1/2} - 2 \cdot T_{i,j,k+1/2} + T_{i-1,j,k+1/2}}{\Delta x^2} + \frac{T_{i,j+1,k} - 2 \cdot T_{i,j,k} + T_{i,j-1,k}}{\Delta y^2} \right) \quad 3.7.77$$

Similarly, the second equation implicit in only the y-direction can be expressed as

$$\frac{T_{i,j,k+1} - T_{i,j,k+1/2}}{\Delta t / 2} = \alpha \cdot \left(\frac{T_{i+1,j,k+1/2} - 2 \cdot T_{i,j,k+1/2} + T_{i-1,j,k+1/2}}{\Delta x^2} + \frac{T_{i,j+1,k+1} - 2 \cdot T_{i,j,k+1} + T_{i,j-1,k+1}}{\Delta y^2} \right) \quad 3.7.78$$

For simplicity, considering $\Delta x = \Delta y$ (square grid), $D_1 = 2/(D+1)$, and $D_2 = 2/(D-1)$, Eq. (3.7.77) becomes

$$-T_{i-1,j,k+1/2} + D_1 \cdot T_{i,j,k+1/2} - T_{i+1,j,k+1/2} = T_{i,j-1,k} + D_2 \cdot T_{i,j,k} + T_{i,j+1,k} \quad 3.7.79$$

and Eq. (3.7.78) becomes

$$-T_{i,j-1,k+1} + D_1 \cdot T_{i,j,k+1} - T_{i,j+1,k+1} = T_{i-1,j,k+1/2} + D_2 \cdot T_{i,j,k+1/2} + T_{i+1,j,k+1/2} \quad 3.7.80$$

The first equation (3.7.79) is solved for $T_{i,j,k+1/2}$. For each value of j , i goes from 2 to n at the half time step. This process is repeated for all the values of j , which also go from 2 to n . Then, Eq. (3.7.80) is solved for $T_{i,j,k+1}$ at the full time interval using the just calculated values of $T_{i,j,k+1/2}$. This time, for each value of i (which goes from 2 to n), j goes from 2 to n . This process is repeated until all the values of $T_{i,j,k+1}$ are calculated at the end of the full time interval.

To have a better understanding of this important method, let us look at an example and solve it step by step. Let us calculate the temperature profile in the cross section shown in Figure 3.7.2.

It is initially at a temperature T_0 . Its boundaries at $x = 0$ and $y = 0$ are insulated (Neumann condition) whereas the boundary at $x = L$ is exposed to free convection at T_{c1} (Robbins condition), and the boundary at $y = L$ is

maintained at temperature T_{c2} (Dirichlet condition). It has been our previous experience in solving the Neumann condition that using the imaginary-node point approach gives a better approximation at the boundary. We will also use this approach here in the case of the two-dimensional problem.

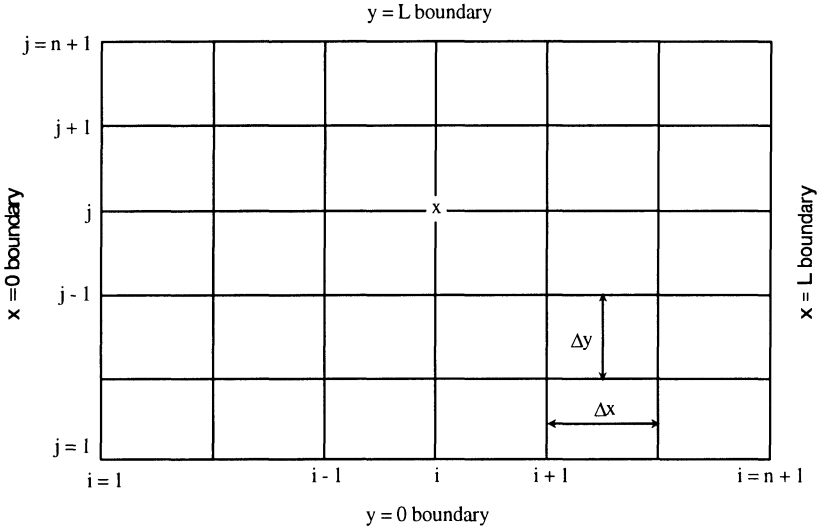


Figure 3.7.2. A finite difference grid in 2-D (X-Y plane).

In Eq. (3.7.79), for $j = 2$, since the boundary at $x = 0$ is insulated, considering an imaginary temperature, and substituting $i = 1$ in Eq. (3.7.79), we obtain a similar expression as Eq.(3.7.52):

$$D_1 \cdot T_{1,j,k+1/2} - 2 \cdot T_{2,j,k+1/2} = T_{1,j-1,k} + D_2 \cdot T_{1,j,k} + T_{1,j+1,k}$$

Substituting in $i = 2$,

$$-T_{1,2,k+1/2} + D_1 \cdot T_{2,2,k+1/2} - T_{3,2,k+1/2} = T_{2,1,k} + D_2 \cdot T_{2,2,k} + T_{2,3,k}$$

where

$T_{1,2,k+1/2}$ is the temperature at $x = 0$ boundary.

Substituting in $i = 3$,

$$-T_{2,2,k+1/2} + D_1 \cdot T_{3,2,k+1/2} - T_{4,2,k+1/2} = T_{3,1,k} + D_2 \cdot T_{3,2,k} + T_{3,3,k}$$

Substituting in $i = n$,

$$\begin{aligned}
 & -T_{n-1,2,k+1/2} + D_1 \cdot T_{n,2,k+1/2} - T_{n+1,2,k+1/2} \\
 & = T_{n,1,k} + D_2 \cdot T_{n,2,k} + T_{n,3,k}
 \end{aligned}$$

where

$T_{n+1,2,k+1/2}$ is the temperature at $x = L$ boundary.

If this boundary is exposed to free convection at T_{c1} , the n -th equation, using the same procedure to arrive at Eq. (3.7.58), becomes

$$-T_{n-1,2,k+1/2} + (D_1 - CC2) \cdot T_{n,2,k+1/2} = T_{n,1,k} + D_2 \cdot T_{n,2,k} + T_{n,3,k} + CC1$$

where

$$CC1 = \frac{T_{c1} \cdot h \cdot \Delta r}{k + h \cdot \Delta r} \quad \text{and} \quad CC2 = \frac{k}{k + h \cdot \Delta r}$$

In matrix form the above set of equations is represented as:

$$\begin{bmatrix}
 D_1 & -2 & & & \\
 -1 & D_1 & -1 & & \\
 & -1 & D_1 & -1 & \\
 \dots & & & & \\
 & & -1 & D_1 & -1 \\
 & & -1 & (D_1 - CC2) &
 \end{bmatrix}
 \begin{bmatrix}
 T_{1,j,k+1/2} \\
 T_{2,j,k+1/2} \\
 T_{3,j,k+1/2} \\
 \dots \\
 T_{n-1,j,k+1/2} \\
 T_{n,j,k+1/2}
 \end{bmatrix}
 =
 \begin{bmatrix}
 R_1 \\
 R_2 \\
 R_3 \\
 \dots \\
 R_{n-1} \\
 R_n + CC1
 \end{bmatrix}
 \quad 3.7.81$$

where for $1 \leq i \leq n-1$ for each j going from 2 to n ,

$$\begin{aligned}
 R_i &= T_{i,j-1,k} + D_2 \cdot T_{i,j,k} + T_{i,j+1,k} \\
 R_n &= R_n + CC1
 \end{aligned}$$

Now, in Eq. (3.7.80), for $i = 2$,

since the side at $y = 0$ is insulated, substituting $j = 1$,

$$D_1 \cdot T_{i,1,k+1} - 2 \cdot T_{i,2,k+1} = T_{i-1,1,k+1/2} + D_2 \cdot T_{i,1,k+1/2} + T_{i+1,1,k+1/2}$$

Substituting in $j = 2$,

$$\begin{aligned}
 & -T_{2,1,k+1} + D_1 \cdot T_{2,2,k+1} - T_{2,3,k+1} \\
 & = T_{1,2,k+1/2} + D_2 \cdot T_{2,2,k+1/2} + T_{3,2,k+1/2}
 \end{aligned}$$

where

$T_{2,1,k+1}$ is the temperature at $y = 0$ boundary.

Substituting in $j = 3$,

$$\begin{aligned} -T_{2,2,k+1} + D_1 \cdot T_{2,3,k+1} - T_{2,4,k+1} \\ = T_{1,3,k+1/2} + D_2 \cdot T_{2,3,k+1/2} + T_{3,3,k+1/2} \end{aligned}$$

.....

Substituting in $j = n$,

$$\begin{aligned} -T_{2,n-1,k+1} + D_1 \cdot T_{2,n,k+1} - T_{2,n+1,k+1} \\ = T_{1,n,k+1/2} + D_2 \cdot T_{2,n,k+1/2} + T_{3,n,k+1/2} \end{aligned}$$

where

$T_{2,n+1,k+1}$ is the temperature at $y = L$ boundary $= T_{c2}$

In matrix form the above set of equations is represented as:

$$\begin{bmatrix} D_1 & -2 & & & \\ -1 & D_1 & -1 & & \\ & -1 & D_1 & -1 & \\ & & & & \\ & & & -1 & D_1 & -1 \\ & & & & -1 & D_1 \end{bmatrix} \begin{bmatrix} T_{i,1,k+1} \\ T_{i,2,k+1} \\ T_{i,3,k+1} \\ \dots\dots\dots \\ T_{i,n-1,k+1} \\ T_{i,n,k+1} \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \dots\dots\dots \\ R_{n-1} \\ R_n + T_{c2} \end{bmatrix} \quad 3.7.82$$

where for $1 \leq j \leq n-1$ for each i going from 2 to n ,

$$\begin{aligned} R_i &= T_{i-1,j,k+1/2} + D_2 \cdot T_{i,j,k+1/2} + T_{i+1,j,k+1/2} \\ R_n &= R_n + T_{c2} \end{aligned}$$

A computer program is developed to solve the above problem using the following information typical to fish or meat products:

Example 3.7.2. A long slab of the product of square cross section (10 cm x 10 cm) having an initial uniform temperature of 5°C , is insulated on its two sides (at $x = 0$ and $y = 0$). The side at $y = L$ is kept in contact with the freezer surface to maintain it at 0°C , whereas the other side at $x = L$ is exposed to the atmosphere of a convective freezer at a constant temperature of -40°C . What will the temperature distribution inside the product be after 1 hour?

$T_0 = 5^\circ\text{C}$; $T_{c1} = -40^\circ\text{C}$; $T_{c2} = 0^\circ\text{C}$
 $L = 10\text{ cm}$; $\rho = 1050\text{ kg/m}^3$; $C_p = 3500\text{ J/kg}\cdot^\circ\text{C}$; $k = 1.5\text{ W/m}\cdot^\circ\text{C}$
 $h = 50\text{ W/m}^2\cdot^\circ\text{C}$ (convective freezer).

Table 3.7.3. Temperature history in a two-dimensional slab (computer output).

Alternating-Direction Implicit Method - two-dimensional parabolic equation					
[X=0 and Y=0:DERIVATIVE; X=L:CONVECTIVE; Y=L:CONST. TEMPERATURE]					
Mean Temperature = -3.06104 at Time = 3600.00000 s					
1.49306	1.42113	1.20121	0.86679	0.49376	0.00000
0.67444	0.63119	0.48275	0.27687	0.10181	0.00000
-1.90581	-1.84287	-1.77666	-1.58400	-1.03109	0.00000
-6.56968	-6.20498	-5.82897	-5.08108	-3.34384	0.00000
-13.86536	-13.05762	-12.47286	-11.35088	-8.33986	0.00000
-23.95629	-23.50548	-23.19940	-22.59413	-20.88250	0.00000

The results are tabulated in Table 3.7.3 in the form of the computer output of the program **ALTDIR.FOR**. The program is listed in Appendix A.3.7.8. With the above information, the geometry was divided into five equal parts ($n = 5$), and a step size of 180 s was chosen because the program calculated its value as 148.75 s considering the stability for an explicit method (Eq. 3.7.74). The bottom row of Table 3.7.3 denotes the side at $x = L$, which is exposed to free convection at -40°C , the last column on its right hand side denotes the side at $y = L$, which has been maintained at 0°C , and the first column and the first row denote the sides at $x = 0$ and $y = 0$, respectively, which are perfectly insulated.

The program is developed in such a way that other initial and boundary conditions can be incorporated with minimum effort. The coefficients of the matrix are calculated by a separate subroutine **COEFF** as in other one-dimensional programs. This matrix is solved by the tridiagonal matrix routine **TRIDGL** developed earlier in Chapter 3.4. Readers are encouraged to use the program for solving problems with different boundary conditions. Only the subroutines **BOUND** and **COEFF** are to be modified. This program can also be generalized by incorporating an unequal step size in x and y in Eqs. (3.7.77) and (3.7.78) and developing general expressions of the type given by Eqs. (3.7.79) and (3.7.80) to take care of unequal steps.

The alternating-direction implicit method is not unconditionally stable, and convergence occurs with a discretization error of the order of $(\Delta t^2 + \Delta x^2)$. It is also a very effective method in solving elliptic PDEs. In solving a three-dimensional problem, the ADI procedure can be used with two intermediate values in each step in time instead of one intermediate value. However, it becomes unstable when $(\alpha \Delta t / \Delta x^2)$ exceeds a certain limit. A modification of the ADI method in light of the Crank-Nicolson concept

makes it a stable procedure for solving three-dimensional parabolic and elliptic problems.

References

1. **Antia, H.M.**, *Numerical Methods for Scientists and Engineers*. Tata McGraw-Hill Publishing Co., Jamshedpur, India, 1991.
2. **Burden, R. L. and Faires, J. D.**, *Numerical Analysis*. Fifth edition, Prindle, Weber & Schmidt, Boston, 1993.
3. **Carnahan, B., Luther, H. A. and Wilkes, J. O.**, *Applied Numerical Methods*. John Wiley & Sons, New York, 1969.
4. **Constantinides, A.**, *Applied Numerical Methods with Personal Computers*. International edition, McGraw-Hill Book Co., New York, 1987.
5. **Hamming, R. W.**, *Numerical Methods for Scientists and Engineers*. Second Edition, International Student Edition, McGraw-Hill, Tokyo, Japan, 1973.
6. **Hornbeck, R. W.**, *Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ, 310pp., 1975.
7. **Patanker, S. V.**, *Computation of Conduction and Duct Flow Heat Transfer*. Innovative Research, Inc., Minnesota, 1991.
8. **Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T.**, *Numerical Recipes — The Art of Scientific Computing*. Cambridge University Press, New York, 818pp., 1988.
9. **Smith, G. D.**, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford Applied Mathematics and Computing Science Series. Clarendon Press – Oxford, New York, 337pp., 1985.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

CHAPTER 3.8

Errors

It has been already discussed that most results obtained by numerical methods are not exact and that the approximate values are accepted within some acceptable error bound. Since errors are inevitable in numerical calculations, it is essential to have some basic concepts about various types of errors that are encountered and ways of estimating them in the light of the topics covered in this text. These errors can be grouped into three categories, namely (i) inherent or experimental, (ii) truncation, and (iii) roundoff.

Any of the three types of error can be expressed in terms of (i) absolute error and (ii) relative error.

Absolute error

As the name suggests, there is a tendency to take it as a positive value. In fact, absolute error may bear any sign and is defined as the difference between the true and the approximate values. In some texts it is defined the other way, that is, the difference between the approximate and the true values. So long as the absolute value of the absolute error is used, this difference in definition does not affect anything. If v_t and v_a are the true and approximate values, respectively, the absolute error, e_a , is given by

$$e_a = v_t - v_a \qquad 3.8.1$$

Relative error

It is defined as the ratio between the absolute error and the true value and generally expressed as a percentage. Thus relative error, e_r , is given as

$$e_r = \frac{e_a}{v_t} = \frac{v_t - v_a}{v_t} = 1 - \frac{v_a}{v_t} \quad 3.8.2$$

When the relative error is expressed as a percentage, the right hand side of Eq. (3.8.2) is multiplied by 100.

For a true value close to unity, the absolute and relative errors are nearly equal, when the latter is expressed as a fraction. Otherwise, one may differ from the other by a large margin.

Inherent error

There are many sources of errors. Since the input data to be processed generally come from experiments, there will be always some **inherent error**, which is caused by unpredictable human or mechanical (or both) mistakes, uncertainty in measurements, oversimplification of the real problem, and approximation of some numbers used in the calculations.

Physical measurements are seldom exact and are generally expressed with some error bound, e.g., a moisture value measured by a moisture meter may be expressed as $(15\% \pm 0.5)$, which says that the moisture content may be within 14.5 and 15.5%. If the measurement is expressed as 14.513559%, it becomes almost obvious that some of the digits after the decimal point are not meaningful because the measurement has not been made with a precision of six significant figures after the decimal.

When a physical quantity is stated without any error bound, it is generally accepted that it is accurate within a half a unit in the last place. A measured moisture content given as 15.27% is considered to be not less than 15.265% and not greater than 15.275%.

In most of the developments of mathematical expressions to simulate a physical process, some assumptions are made without which the derivation becomes cumbersome or impossible. Such assumptions are mostly simplified versions of most complicated physical or chemical or biological phenomena for which no mathematical explanation exists. Mathematical models, thus developed, cannot properly explain the complicated phenomenon for which a simplified assumption has been made, and an inherent error is imbedded in the simulation.

The constant **pi** (π), which relates the diameter of a circle to its circumference, is a transcendental number and does not have an exact finite decimal representation. In many calculations this number is used and thus an inherent error becomes inevitable in the results. In this century, humans have succeeded reaching the moon. Supercomputers with extremely high precision

are being used to calculate the trajectory of a satellite. In these calculations, maximum effort is being made to minimize the inherent error. Thus the value of π should be used up to possibly 40 significant digits after the decimal whereas, in our day-to-day use, a value of 3.14159265 looks perfectly reasonable. If this value were used in the satellite trajectory calculations, certainly the satellite would have missed the moon by more than a hair!

Truncation error

The truncation error is caused by the numerical method itself, which is derived employing one or more infinite series. Since physically it is not possible to include all the terms of the infinite series, approximation is made by considering only a few first terms and thus giving rise to an inevitable truncation error.

The example of the Taylor Series expansion of e^x about $x = 0$, and finding the values of $e^{0.01}$ and $e^{1.01}$ considering only the first four terms, given in Chapter 3.1, can be again cited to explain the truncation error. When the value of x is small, the truncation error is also very small, but when x is large the truncation error becomes so large that the result looks meaningless.

Before explaining the third type of error, namely, the roundoff error, it is necessary to explain the term **significant figures**.

In a floating point system, the numbers are expressed by a fixed number of significant digits. Significant figures of a floating number are any given digits of the number except for zeros to the left of a nonzero digit just meant for positioning the decimal point. The numbers 15.2750, 152750, 0.152750, and 0.00152750 all have six significant digits.

When two very small and nearly equal numbers that have the smallest exponent that a particular computer can handle are subtracted from each other, the subtracted value gets an exponent that the computer cannot represent with its limited capacity. This gives rise to a situation called **floating point underflow**. Commonly, a computer handles this situation by attributing a value equal to zero to the result of the subtraction. This also can happen when the smallest number is divided by a number larger than unity.

Similarly, when two very large numbers, each bearing the maximum size of exponent that a particular computer can handle, are added together, the value of the sum gets an exponent beyond the capacity of that computer, and, as a result, **floating point overflow** occurs. It also may occur when the largest number is divided by a fraction. Most compilers handle such a situation by interrupting the execution of program and giving a **Real Math Overflow** message.

Roundoff error

The conventional process of rounding a number to n significant figures consists of replacing that number by an n -digit approximation with minimum

error. The process of rounding off should clearly be distinguished from the process called **chopping off**, which occurs when an infinite series is truncated after some initial terms. Thus chopping is a process of discarding all figures following the n -th digit without modifying the n -th digit. The rule of rounding off, on the other hand, is as follows.

If a floating point number is to be rounded off to n significant figures, the $(n+1)$ -th and subsequent figures are discarded. When the discarded number is less than a half a unit in the $(n+1)$ -th place, the n -th figure of the number to be rounded off should be kept unchanged. On the other hand, if it is greater than half a unit in the $(n+1)$ -th place, one is added to the n -th figure of the number being rounded. If it is exactly half a unit, the n -th figure of the number should be rounded off to the nearest even figure.

When the numbers 15.2755 and 15.2745 are rounded off to three decimal places, they become 15.276 and 15.274, respectively. If the number 15.27545 is rounded off up to one significant figure, we get 15.2754, 15.275, 15.28, 15.3, and 15.

The roundoff error is propagative but not transitive and is dangerous when there are many-step calculations. If the proper measure is not taken, it can ruin a computation completely. One way to combat this error is to take extra significant figures in the calculations. The double precision arithmetic helps in this matter. The other appropriate method is to scale the data so that the number of significant figures is reduced. A classic example of this measure was explained in Chapter 3.5, where the input values of the independent variable are divided by the largest number to reduce roundoff error in the least squares approximation for higher-order polynomials.

The following steps will help reduce error in general:

- Step 1. The subtraction of two nearly equal and small numbers should be avoided by all means. It should be noted that the Secant method of root finding suffers this defect. Perhaps this was the reason why it did not work for finding the multiple root in Chapter 3.2,
- Step 2. The number of arithmetic operations should be minimized,
- Step 3. Allow extra significant figures in the calculations,
- Step 4. Scaling up of data, if possible, which have more number of significant figures than necessary.

Error estimation

The first chapter on numerical methods is finite difference calculus, and we will start there. In developing the finite difference expressions for first and higher derivatives of a function, Taylor Series expansion with limited terms has been used. By selecting some finite terms and neglecting the rest, all the expressions have suffered the truncation error. One general example with a finite Taylor Series is used to illustrate the truncation error that the methods that use some finite terms of such infinite series suffer with.

If a function $f(x)$ having continuous $n + 1$ derivatives is expanded about $x = a$ by the Finite Taylor Series, we get

$$f(x) = f(a) + (x-a).f'(a) + \frac{(x-a)^2}{2!}.f''(a) + \dots + \frac{(x-a)^n}{n!}.f^{(n)}(a) + \frac{(x-a)^{n+1}}{(n+1)!}.f^{(n+1)}(\xi) \quad 3.8.3$$

where ξ is some number between a and x .

On the other hand, if $f(x)$ is approximated by the first $n+1$ terms, the truncation error is represented by the remainder term of Eq. (3.8.3), which is

$$\frac{(x-a)^{n+1}}{(n+1)!}.f^{(n+1)}(\xi)$$

The example to demonstrate this error is taken from chapter 3.1 as the five-term Taylor Series expansion of e^x about $a = 0$, which is

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

Using Eq. (3.8.3), the remainder term, which is equal to the truncation error, e_{tr} , can be expressed as

$$e_{tr} = \frac{x^5}{5!}.e^{\xi} \quad (\xi \text{ between } 0 \text{ and } x)$$

Thus the error e_{tr} is smaller than $x^5/120$, if e is a fraction, which is the case as we have observed in Chapter 3.1 that for x greater than unity the approximation does not work. If x is taken as 0.1, the error should be between 0 and $1/12,000,000$, that is, it is smaller than 8.3×10^{-8} . The value of $e^{0.1}$, considering the first five terms, is 1.10517083, where each term was taken up to eight significant figures. If the result is rounded off to four significant digits after the decimal point, it becomes 1.1052, and the roundoff error with respect to the exact value up to eight significant figures after the decimal turns out to be

$$\begin{aligned} e_a &= 1.10517092 - 1.1052 \\ &= -2.908 \times 10^{-5} \end{aligned}$$

It is interesting to note that, if each of the Taylor Series terms is rounded off up to four significant figures after the decimal, the result suffers the same roundoff error as the one just calculated. If four additional digits are retained in each term, the roundoff error is reduced to 9.0×10^{-8} . This illustrates the fact that double precision arithmetic helps reduce this error.

References

1. **Dorn, W. S. and McCracken, D. D.**, *Numerical Methods with FORTRAN IV Case Studies*. John Wiley & Sons, New York, 1972.
2. **Hamming, R.W.**, *Numerical Methods for Scientists and Engineers*. Second Edition, International Student Edition, McGraw-Hill, Tokyo, Japan, 1973.
3. **Hildebrand, F. B.**, *Introduction to Numerical Analysis*. Second Edition, Dover Publications, New York, 1987.
4. **Hornbeck, R. W.**, *Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ, 310pp., 1975.
5. **Kreyszig, E.**, *Advanced Engineering Mathematics*. Seventh edition, John Wiley & Sons, New York, 1993.

CHAPTER 3.9

Finite Element Methods

In the discussion of finite difference methods, we observed that the differential equations were approximated by replacing the derivatives with difference quotients. These quotients contained the values of the solution at certain discrete points that were identified by using a mesh. The equations obtained by this procedure were solved to obtain the unknown values at the nodal points. The application of finite difference methods becomes difficult when the geometrical shape is complex with nonlinear boundaries, and the properties of the material are nonuniform. For these complicated situations, the finite element method offers a simple procedure to solve for the unknown quantities.

The finite element method is relatively new yet it is now widely used for a broad range of applications in engineering and mathematical physics. The initial applications of this method, in the 1950s, were limited to the aerospace industry. Now it has become the method of choice in solving problems in structural mechanics, solid mechanics, fluid mechanics, and heat transfer. This method offers an efficient computational procedure useful in interpolating the approximate solution to boundary-value problems.

In this chapter, a brief introduction to the finite element method is given. A complete treatment of this topic is beyond the scope of this book. A number of references are provided in the references of this chapter for the interested readers to consult.

From a broad perspective, the finite element method involves taking a problem that is fairly large in its scope and discretizing it into a large number of small problems. These small problems are then assembled and solved to yield a large number of solutions that are then combined to give the solution to the original problem. This procedure implies that the successful execution of this method requires the use of high-speed computers.

More specifically, the finite element method involves taking a domain that is geometrically complex and representing it with a number of subdomains that are geometrically simple. These subdomains are called finite elements. The main concept behind the finite element method is that any continuous function can be represented by a linear combination of algebraic polynomials. To meet this objective, for each finite element, an approximation function is derived. These approximation functions also referred to as interpolation functions provide the element equations. These element equations are then assembled, and, using the convergence procedures, the final solution is obtained.

Procedure Steps in Using Finite Element Methods

The procedural steps used in the finite element method is summarized in the following:

Discretization of the Domain

A given domain is represented by a collection of a finite number of subdomains called elements. The elements may be one-dimensional straight lines, or two-dimensional triangles or rectangles, or even three-dimensional regions. The nodes of the elements are defined by numbers. A number of elements in the domain are connected by common nodes. The finite element mesh obtained from the discretization step may be uniform if all the elements are of the same size, or it may be nonuniform to accommodate complex geometrical shapes.

Development of Element Equations

For a typical element, the governing equation is often a differential equation. Since an exact solution for the differential equation may not be possible, a continuous function with unknown coefficients is used as an approximation of the solution for each element. The approximating function is also referred to as the interpolation function. The unknown coefficients in the interpolation function are then evaluated using procedures such as variational methods and the method of weighted residuals.

Assembly of Element Equations for the Domain

The element equations obtained from the preceding step are linked together such that the continuity between the elements is maintained. This step assures that the solution is continuous for the entire domain. A matrix formulation is used to incorporate all relevant equations. Next, the boundary and initial conditions are introduced.

Solution

Finally, the assembled equations for all the elements with boundary and initial conditions are solved using matrix solving techniques to obtain the final solution.

In the following sections we will consider some of the preceding steps in more detail with examples to illustrate the procedures.

Properties of Elements

In the finite element method, the element plays an important role in the solution of the problem. Therefore, it is important to understand the properties of a single element.

Mathematical functions are used to approximate the unknown parameters over the domain of an element. These unknown parameters may be quantities such as temperature, pressure, force, or displacement. Among various functions, the most commonly used function is a polynomial. The number of nodes and the number of unknowns for each of the nodes in an element influence the order of the polynomial.

One-dimensional Simplex Element

Let us first consider a one-dimensional element. Such an element, called a **simplex element** has a line segment and two nodes at the ends. Let the nodal values be represented by two scalar quantities, namely, Φ_i and Φ_j . In Figure 3.9.1, a simplex element is shown. Note that the origin of the coordinate system lies outside the element.

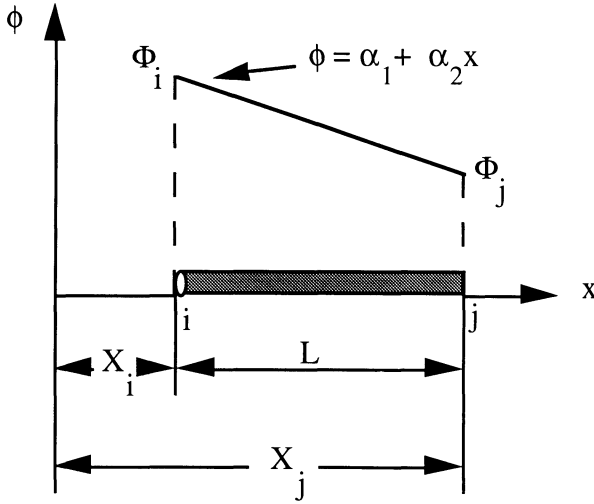


Figure 3.9.1 A simplex one-dimensional element.

The polynomial function for f , a scalar quantity, is

$$\phi = \alpha_1 + \alpha_2 x \quad 3.9.1$$

As shown in Figure 3.9.1 the values of scalar quantity ϕ are given as follows

$$\begin{aligned} \phi &= \Phi_i \quad \text{at } x = X_i \\ \phi &= \Phi_j \quad \text{at } x = X_j \end{aligned}$$

Thus,

$$\phi = \left(\frac{\Phi_i X_j - \Phi_j X_i}{L} \right) + \left(\frac{\Phi_j - \Phi_i}{L} \right) x \quad 3.9.2$$

where

$$L = X_j - X_i$$

Eq. (3.9.2) may be rearranged to give

$$\phi = \left(\frac{X_j - x}{L} \right) \Phi_i + \left(\frac{x - X_i}{L} \right) \Phi_j \quad 3.9.3$$

The quantities enclosed in parenthesis in the above equation are called the shape functions. The shape functions N_i and N_j are:

$$N_i = \frac{X_j - x}{L} \quad 3.9.4$$

$$N_j = \frac{x - X_i}{L} \quad 3.9.5$$

Eq. (3.9.3) can be written in matrix notation for ease of solving on a computer. Thus

$$\phi = [N]\{\Phi\} \quad 3.9.6$$

where

$$[N] = [N_i \quad N_j] \quad 3.9.7$$

$$\{\Phi\} = \begin{Bmatrix} \Phi_i \\ \Phi_j \end{Bmatrix} \quad 3.9.8$$

It is important to observe two properties of shape functions. First, the value of N_i is zero at node j ; similarly, N_j is zero at node i . Second, the value of N_i is 1 at node i , similarly N_j is 1 at node j .

Two-dimensional simplex element:

A triangular element is a two-dimensional element. If we assume that the scalar quantity, ϕ varies linearly, then the interpolation polynomial can be written as:

$$\phi = N_i \Phi_i + N_j \Phi_j + N_k \Phi_k \quad 3.9.9$$

The equations of shape functions N_i , N_j , and N_k can be derived using the procedures given by Segerlind (1984).

$$N_i = \frac{1}{2A} (a_i + b_i x + c_i y)$$

where

$$a_i = X_j Y_k - X_k Y_j \quad 3.9.10$$

$$b_i = Y_j - Y_k$$

$$c_i = X_k - X_j$$

$$N_j = \frac{1}{2A}(a_j + b_jx + c_jy)$$

where

$$a_j = X_k Y_i - X_i Y_k \quad 3.9.11$$

$$b_j = Y_k - Y_i$$

$$c_j = X_i - X_k$$

and

$$N_k = \frac{1}{2A}(a_k + b_kx + c_ky)$$

where

$$a_k = X_i Y_j - X_j Y_i \quad 3.9.12$$

$$b_k = Y_i - Y_j$$

$$c_k = X_j - X_i$$

where

$$2A = \begin{vmatrix} 1 & X_i & Y_i \\ 1 & X_j & Y_j \\ 1 & X_k & Y_k \end{vmatrix} \quad 3.9.13$$

The shape functions, N_i , N_j , and N_k , have unique properties, such as, N_i having a value of one at node i but zero at nodes j and k . The shape function, N_j , has a value of one at node j , but zero at nodes i and k . Similarly, the shape function, N_k , has a value of one at node k but zero at nodes i and j . It should be noted that, since we assume that the gradient of ϕ is constant over an element, whenever high gradients are involved then we must use several small size elements to approximate such gradients. A continuity exists between the values of ϕ between the boundary of one element with another.

Example 3.9.1: Consider temperature distribution along a stainless steel fin. If a linear element is chosen such that node i (2 cm from the origin) is at 40°C, and node j (10 cm from the origin) is at 60°C. Determine the temperature at a location 8 cm from the origin of the coordinate system.

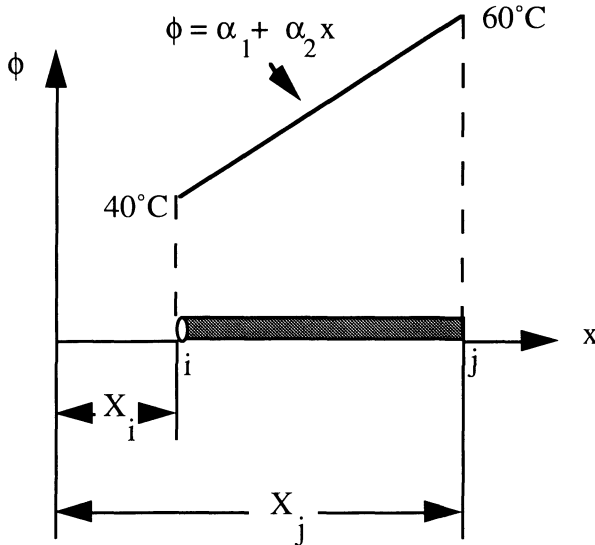


Figure 3.9.2 A one-dimensional element

Referring to Figure 3.9.2, we get

$$\begin{array}{ll} X_i = 2 \text{ cm} & L = (10-2) = 8 \text{ cm} \\ X_j = 10 \text{ cm} & \Phi_i = 40^\circ\text{C} \\ x = 8 \text{ cm} & \Phi_j = 60^\circ\text{C} \end{array}$$

Therefore,

$$\begin{aligned} \phi &= \left(\frac{10-8}{8} \right) 40 + \left(\frac{8-2}{8} \right) 60 \\ &= \left(\frac{2}{8} \right) 40 + \left(\frac{6}{8} \right) 60 \\ &= 10 + 45 \\ &= 55^\circ\text{C} \end{aligned}$$

Thus the temperature at a point 8 cm away from the origin is 55°C.

Example 3.9.2: Consider a two-dimensional triangular element as shown in Figure 3.9.3. The nodal temperatures are 60°C at node $(0,0)$; 30°C at node $(3,2)$; and 35°C at node $(1,5)$. We want to determine temperature at location $A(2,2)$.

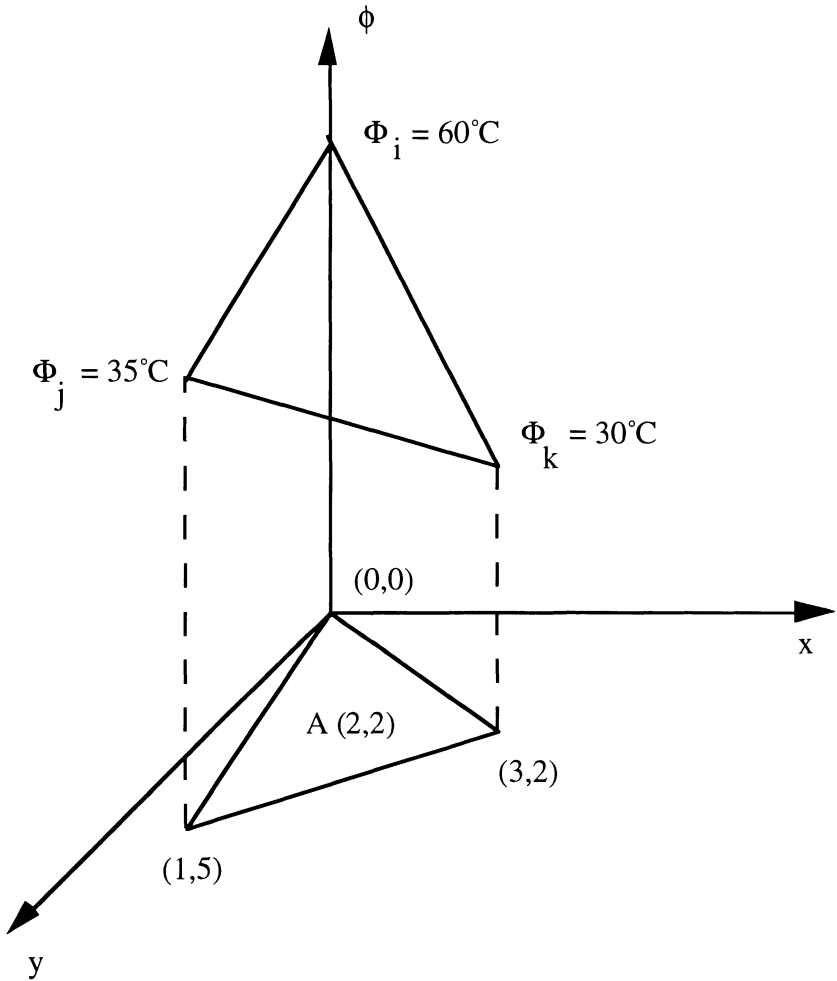


Figure 3.9.3 A two dimensional triangular element.

First, we calculate the shape functions, using Eqs. (3.9.10), (3.9.11), and (3.9.12)

$$\begin{aligned}
 a_i &= X_j Y_k - X_k Y_j = (1)(2) - (3)(5) = -13 \\
 b_i &= Y_j - Y_k = (5) - (2) = 3 \\
 c_i &= X_k - X_j = (3) - (1) = 2 \\
 \\
 a_j &= X_k Y_i - X_i Y_k = (3)(0) - (0)(2) = 0 \\
 b_j &= Y_k - Y_i = 2 - 0 = 2 \\
 c_j &= X_i - X_k = 0 - 3 = -3 \\
 \\
 a_k &= X_i Y_j - X_j Y_i = (0)(5) - (1)(0) = 0 \\
 b_k &= Y_i - Y_j = (0) - (5) = -5 \\
 c_k &= X_j - X_i = (1) - (0) = 1
 \end{aligned}$$

$$2A = \begin{vmatrix} 1 & X_i & Y_i \\ 1 & X_j & Y_j \\ 1 & X_k & Y_k \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 5 \\ 1 & 3 & 2 \end{vmatrix}$$

$$2A = 1(2 - 15) = -13$$

Therefore,

$$N_i = \frac{1}{2A} [a_i + b_i x + c_i y]$$

$$= \frac{1}{-13} [-13 + 3x + 2y]$$

$$N_j = \frac{1}{2A} [a_j + b_j x + c_j y]$$

$$= \frac{1}{-13} [0 + 2x - 3y]$$

$$N_k = \frac{1}{2A} [a_k + b_k x + c_k y]$$

$$= \frac{1}{-13} [0 - 5x + y]$$

The approximating function for the triangular element can be written using Eq. (3.9.9) as follows:

$$\phi = \frac{(-13 + 3x + 2y)}{-13} \Phi_i + \frac{(2x - 3y)}{-13} \Phi_j + \frac{(-5x + y)}{-13} \Phi_k$$

Therefore, temperature at location A (2, 2) is obtained by substituting $x=2$ and $y=2$ in the preceding expression. Thus

$$\phi = \left[\left(\frac{-3}{-13} \right) 60 \right] + \left[\left(\frac{-2}{-13} \right) 35 \right] + \left[\left(\frac{-8}{-13} \right) 30 \right]$$

The temperature at A (2,2) = 37.69°C

Embedding Element Equations into a Domain

As an illustration of how element equations are developed for a domain, let us consider a one-dimensional case of a long thin rod that has fixed boundary conditions and a continuous heat source along its axis (Figure 3.9.4).

We can describe this situation with the following equation:

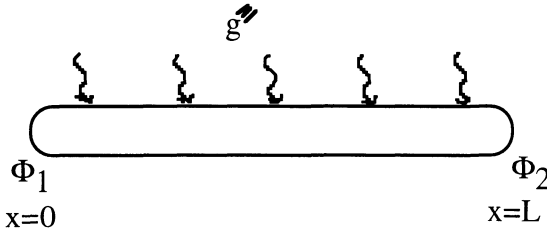


Figure 3.9.4. A one-dimensional rod with a uniform heat source.

$$\frac{d^2\phi}{dx^2} + \frac{g'''}{k} = 0 \quad 3.9.14$$

where g''' denotes the heat source along the rod per unit volume per unit time (J/sm^3), and k is the thermal conductivity of the material (W/mC). The fixed boundary conditions are

$$\begin{aligned} \phi(0) &= \Phi_1 \\ \phi(L) &= \Phi_2 \end{aligned} \quad 3.9.15$$

The temperature distribution within the element can be described by using the following expression

$$\phi = N_1 \Phi_1 + N_2 \Phi_2 \quad 3.9.16$$

The element equations will be obtained using the method of weighted residuals. Rewrite Eq. (3.9.14) as

$$0 = \frac{d^2\phi}{dx^2} + \frac{g'''}{k} \quad 3.9.17$$

Next, substituting the approximate solution from Eq. (3.9.16) into Eq. (3.9.17), then due to the approximation we will not obtain an exact solution, but our solution will contain a residual value, R , where

$$R = \frac{d^2\tilde{\phi}}{dx^2} + \frac{g'''}{k} \quad 3.9.18$$

The weighted residuals method involves finding a minimum for the residual, R , in a general expression:

$$\int_S W_i R dS = 0 \quad \text{for } i = 1, 2, \dots, p \quad 3.9.19$$

where S is the entire domain, and W_i are the independent weighting functions. The most commonly used procedure to carry out the minimization process is called the **Galerkin method**. This method uses the shape functions, N_i , for the weighting functions, thus

$$\int_S N_i R dS = 0 \quad i = 1, 2, 3, \dots, p \quad 3.9.20$$

The use of the Galerkin method is illustrated using our example of the one-dimensional rod. Substituting Eq. (3.9.18) in Eq. (3.9.20), we get,

$$\int_{x_1}^{x_2} N_i \left[\frac{d^2\tilde{\phi}}{dx^2} + \frac{g'''}{k} \right] dx = 0 \quad i = 1, 2 \quad 3.9.21$$

$$\int_{x_1}^{x_2} N_i(x) \frac{d^2\tilde{\phi}}{dx^2} dx = - \int_{x_1}^{x_2} N_i(x) \frac{g'''}{k} dx \quad i = 1, 2 \quad 3.9.22$$

The left hand side of Eq. (3.9.22) may be expanded by using the procedure of integration by parts:

$$\int_{x_1}^{x_2} N_i(x) \frac{d^2 \tilde{\phi}}{dx^2} dx = N_i(x) \frac{d\tilde{\phi}}{dx} \Big|_{x_1}^{x_2} - \int_{x_1}^{x_2} \frac{d\tilde{\phi}}{dx} \frac{dN_i}{dx} dx \quad i = 1, 2 \quad 3.9.23$$

Using the properties of shape functions, since $N_1(x_2) = 0$ and $N_1(x_1) = 1$, the terms on the left hand side of Eq. (3.9.23) are reduced to the following:

for $i=1$

$$-\frac{d\phi(x_1)}{dx} \quad 3.9.24$$

and, for $i=2$

$$\frac{d\phi(x_2)}{dx} \quad 3.9.25$$

The preceding two terms represent the boundary conditions for this problem.

We can rewrite Eq. (3.9.23), for $i=1$:

$$\int_{x_1}^{x_2} \frac{d\tilde{\phi}}{dx} \frac{dN_1}{dx} dx = -\frac{d\phi(x_1)}{dx} + \int_{x_1}^{x_2} \frac{g'''}{k} N_1(x) dx \quad 3.9.26$$

and for $i=2$:

$$\int_{x_1}^{x_2} \frac{d\tilde{\phi}}{dx} \frac{dN_2}{dx} dx = \frac{d\phi(x_2)}{dx} + \int_{x_1}^{x_2} \frac{g'''}{k} N_2(x) dx \quad 3.9.27$$

In Eqs. (3.9.26) and (3.9.27), the left hand side terms can be expanded by substituting the derivatives of N_1 and N_2 . These derivatives are obtained by differentiating Eqs. 3.9.4 and 3.9.5, as follows

Since

$$N_1 = \frac{x_2 - x}{x_2 - x_1} \quad 3.9.28$$

therefore,

$$\frac{dN_1}{dx} = -\frac{1}{x_2 - x_1} \quad 3.9.29$$

and, since

$$N_2 = \frac{x - x_1}{x_2 - x_1} \quad 3.9.30$$

therefore,

$$\frac{dN_2}{dx} = \frac{1}{x_2 - x_1} \quad 3.9.31$$

Also, differentiating Eq. 3.9.16, we get

$$\frac{d\phi}{dx} = \frac{dN_1}{dx} \Phi_1 + \frac{dN_2}{dx} \Phi_2 \quad 3.9.32$$

then substituting the respective quantities in the preceding equation,

$$\frac{d\phi}{dx} = \frac{(-\Phi_1 + \Phi_2)}{x_2 - x_1} \quad 3.9.33$$

By substituting Eqs. (3.9.29), (3.9.31), and (3.9.33) into the left hand side terms of Eqs. (3.9.26) and (3.9.27), we get,

for $i=1$:

$$\int_{x_1}^{x_2} \frac{d\tilde{\phi}}{dx} \frac{dN_1}{dx} dx = \int_{x_1}^{x_2} \frac{\Phi_1 - \Phi_2}{(x_2 - x_1)^2} dx = \frac{(\Phi_1 - \Phi_2)}{(x_2 - x_1)} \quad 3.9.34$$

for $i=2$:

$$\int_{x_1}^{x_2} \frac{d\tilde{\phi}}{dx} \frac{dN_2}{dx} dx = \int_{x_1}^{x_2} \frac{-\Phi_1 + \Phi_2}{(x_2 - x_1)^2} dx = \frac{(-\Phi_1 + \Phi_2)}{(x_2 - x_1)} \quad 3.9.35$$

Then, combining these results, we obtain the element equations written in matrix notation as:

$$\frac{1}{x_2 - x_1} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \end{Bmatrix} = \begin{Bmatrix} -\frac{d\Phi(x_1)}{dx} \\ \frac{d\Phi(x_2)}{dx} \end{Bmatrix} + \begin{Bmatrix} \int_{x_1}^{x_2} \frac{g'''}{k} N_1(x) dx \\ \int_{x_1}^{x_2} \frac{g'''}{k} N_2(x) dx \end{Bmatrix} \quad 3.9.36$$

In Eq. (3.9.36) the left hand side terms are called the **Element stiffness matrix**, the first term on the right hand side represents the **boundary conditions** and the second term accounts for the **external effects**.

At this point it would be instructive to use numerical data for the example of heat transfer along the rod. Let us assume that the heat source term is 90 W/m^3 , and thermal conductivity is $15 \text{ W/m}^\circ\text{C}$. The boundary conditions are $T(0,t) = 25^\circ\text{C}$ and $T(20,t) = 100^\circ\text{C}$.

From the given data, $\frac{g'''}{k} = 6^\circ\text{C/m}^2$. We will divide the 15 cm long rod into five elements, where the length of each element is 3 cm.

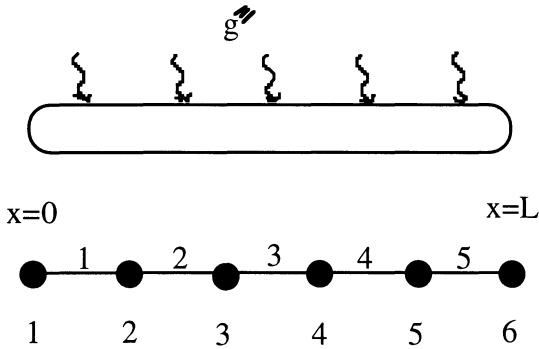


Figure 3.9.5 A one-dimensional rod divided into five linear elements.

To obtain the element equation for this example, we can use Eqs. (3.9.4) and (3.9.5), where the individual terms are evaluated as follows:

$$N_1 = \frac{3-x}{3}$$

Then,

$$\int_0^3 6 \frac{3-x}{3} dx = 9 \quad 3.9.37$$

Similarly, $N_2 = \frac{x-0}{3}$, then,

$$\int_0^3 6 \frac{x-0}{3} dx = 9 \quad 3.9.38$$

Substituting the above results into Eq. (3.9.36), we get,

$$0.33\Phi_1 - 0.33\Phi_2 = -\frac{d\phi(x_1)}{dx} + 9 \quad 3.9.39$$

and

$$-0.33\Phi_1 + 0.33\Phi_2 = \frac{d\phi(x_2)}{dx} + 9 \quad 3.9.40$$

The next step is to develop a global numbering system. This step is rather easy for a one-dimensional scheme, but it becomes more involved for two- or three- dimensional elements. Thus, for the five elements shown in Figure 3.9.5, we can generate the following table.

Element	Local Node Numbers	Global Node Numbers
1	1	1
	2	2
2	1	2
	2	3
3	1	3
	2	4
4	1	4
	2	5
5	1	5
	2	6

Using the global coordinates, we can write element equations for the entire domain. We will do this in matrix notation. Thus, we have the first equation as

$$\begin{bmatrix} 0.33 & -0.33 & 0 & 0 & 0 & 0 \\ -0.33 & 0.33 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} \frac{-d\Phi(x_1)}{dx} + 9 \\ \frac{d\Phi(x_2)}{dx} + 9 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

The second and remaining equations are as follows:

$$\begin{bmatrix} 0.33 & -0.33 & 0 & 0 & 0 & 0 \\ -0.33 & 0.33 + 0.33 & -0.33 & 0 & 0 & 0 \\ 0 & -0.33 & 0.33 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ 0 \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} \frac{-d\Phi(x_1)}{dx} + 9 \\ \frac{dx}{9} + 9 \\ \frac{d\Phi(x_3)}{dx} + 9 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

$$\begin{bmatrix} 0.33 & -0.33 & 0 & 0 & 0 & 0 \\ -0.33 & 0.66 & -0.33 & 0 & 0 & 0 \\ 0 & -0.33 & 0.33 + 0.33 & -0.33 & 0 & 0 \\ 0 & 0 & -0.33 & 0.33 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} \frac{-d\Phi(x_1)}{dx} + 9 \\ 18 \\ 9 + 9 \\ \frac{d\Phi(x_4)}{dx} + 9 \\ 0 \\ 0 \end{Bmatrix}$$

$$\begin{bmatrix} 0.33 & -0.33 & 0 & 0 & 0 & 0 \\ -0.33 & 0.66 & -0.33 & 0 & 0 & 0 \\ 0 & -0.33 & 0.66 & -0.33 & 0 & 0 \\ 0 & 0 & -0.33 & 0.33 + 0.33 & -0.33 & 0 \\ 0 & 0 & 0 & -0.33 & 0.33 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ 0 \end{Bmatrix} = \begin{Bmatrix} \frac{-d\Phi(x_1)}{dx} + 9 \\ 18 \\ 18 \\ 9 + 9 \\ \frac{d\Phi(x_5)}{dx} + 9 \\ 0 \end{Bmatrix}$$

$$\begin{bmatrix} 0.33 & -0.33 & 0 & 0 & 0 & 0 \\ -0.33 & 0.66 & -0.33 & 0 & 0 & 0 \\ 0 & -0.33 & 0.66 & -0.33 & 0 & 0 \\ 0 & 0 & -0.33 & 0.66 & -0.33 & 0 \\ 0 & 0 & 0 & -0.33 & 0.33 + 0.33 & -0.33 \\ 0 & 0 & 0 & 0 & -0.33 & 0.33 \end{bmatrix} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \end{Bmatrix} = \begin{Bmatrix} \frac{-d\Phi(x_1)}{dx} + 9 \\ 18 \\ 18 \\ 18 \\ 9 + 9 \\ \frac{d\Phi(x_6)}{dx} + 9 \end{Bmatrix}$$

$$\begin{bmatrix} 0.33 & -0.33 & 0 & 0 & 0 & 0 \\ -0.33 & 0.66 & -0.33 & 0 & 0 & 0 \\ 0 & -0.33 & 0.66 & -0.33 & 0 & 0 \\ 0 & 0 & -0.33 & 0.66 & -0.33 & 0 \\ 0 & 0 & 0 & -0.33 & 0.66 & -0.33 \\ 0 & 0 & 0 & 0 & -0.33 & 0.33 \end{bmatrix} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \end{Bmatrix} = \begin{Bmatrix} \frac{-d\Phi(x_1)}{dx} + 9 \\ 18 \\ 18 \\ 18 \\ 18 \\ \frac{d\Phi(x_6)}{dx} + 9 \end{Bmatrix}$$

The matrix in the preceding equation can be solved to obtain the unknown values of nodal temperatures and the flux at nodes 1 and 6.

Thus

$$\begin{aligned}
 \frac{d\Phi(x_1)}{dx} - 0.33\Phi_2 &= 0.75 \\
 0.66\Phi_2 - 0.33\Phi_3 &= 26.25 \\
 -0.33\Phi_2 + 0.66\Phi_3 - 0.33\Phi_4 &= 18.00 \\
 -0.33\Phi_3 + 0.66\Phi_4 - 0.33\Phi_5 &= 18.00 \\
 -0.33\Phi_4 + 0.66\Phi_5 &= 51.00 \\
 -0.33\Phi_5 - \frac{d\Phi(x_6)}{dx} &= -24.00
 \end{aligned}$$

Solving the above simultaneous equations, we obtain

$$\begin{aligned}
 \frac{d\Phi(x_1)}{dx} &= 49.95 \\
 \Phi_2 &= 149.1^\circ\text{C} \\
 \Phi_3 &= 218.6^\circ\text{C} \\
 \Phi_4 &= 233.6^\circ\text{C} \\
 \Phi_5 &= 194^\circ\text{C} \\
 \frac{d\Phi(x_6)}{dx} &= -40.05
 \end{aligned}$$

The exact solution for Eq. (3.9.14) is

$$\frac{\Phi - \Phi_b}{\Phi_a - \Phi_b} = \left(1 - \frac{x}{L}\right) \left(\frac{g'''L^2x}{2k(\Phi_a - \Phi_b)L} + 1 \right) \quad 3.9.41$$

where Φ_a and Φ_b are the boundary temperatures. Using the exact solution, we obtain

$$\begin{aligned}
 \Phi_2 &= 148^\circ\text{C} \\
 \Phi_3 &= 217^\circ\text{C} \\
 \Phi_4 &= 232^\circ\text{C} \\
 \Phi_5 &= 193^\circ\text{C}
 \end{aligned}$$

The results obtained from using the finite element method compare favorably with those calculated from the exact solution.

The above procedure included all the major steps in solving problems using the finite element method. We used a simple example, and a one-dimensional case to illustrate the method. The complexities increase as one considers more complicated geometrical shapes and two- or three-dimensional domains. However, the procedural steps remain the same as

seen in this example. Finite element methods are expected to continue to gain in popularity due to their key advantages such as that these methods can easily accommodate geometric irregularities, the element size can be easily varied, and the methods are easily adapted when material discontinuities are present.

References

1. **Chapra, S.C. and R.P. Canale.** *Numerical Methods for Engineers.* McGraw Hill Book Company. New York, 1988.
2. **Desai, C.S.** *Elementary Finite Element Method.* Prentice-Hall, Englewood Cliffs, NJ, 1979 .
3. **Reddy, J. N.** *An Introduction to the Finite Element Method.* McGraw Hill Book Company. New York, 1984.
4. **Segerlind, L.J.** *Applied Finite Element Analysis.* John Wiley and Sons. New York, 1984.
5. **Zienkiewicz, O.C.** *The Finite Element Method.* McGraw Hill Book Company, London, 1977.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

PART IV

DIMENSIONAL ANALYSIS

In many an occasion the task of fitting experimental data to appropriate design equations becomes complicated even with the help of advanced regression packages. In such a case where a complete mathematical treatment is not possible, the method used to develop mathematical models is based on the concept of dimensions (see Part I) and the use of dimensional formulas. This method is precisely called **dimensional analysis** which is an alternative method between a pure mathematical development and a completely empirical study. It is based on the fact that if the equation correlating all the pertinent variables in a physical process exists, it must be dimensionally homogeneous. Besides mathematical modeling, its use also includes checking the consistency of the units in equations, converting data from one system of units to another as discussed in Part I, and scaling up a process or an equipment to estimate the performance of the full-scale model. For no apparent reason, its application in the field of agricultural and food engineering has been limited. Mostly in the area of fluid mechanics, the technique of dimensional analysis has extensively been applied.

For dimensional analysis, physical quantities are divided into two groups — the primary group and the secondary group — very similar to independent and dependent variables in the case of regressions. A small number of physical quantities come in the first group, which are chosen first. Then the remainder in the second group is expressed in terms of the first one.

The choice of either quantity does not go by any established rules. It is quite arbitrary and depends very much on the experience, discretion, and convenience of the particular investigator in question. For all engineering areas, the basic dimensions such as mass, length, time, temperature, and electric charge may be a satisfactory list of the primary quantities though

these are not to be confused with basic units. In unit operations, electric charge may not be required. In the list of primary quantities, sometimes it is advantageous to add force and heat.

Between any two units in the set defining a primary quantity, there should be a conversion factor relating the two units. If [L] is the dimension of length, the set designating all the units of [L] can be expressed as

$$L = \{\text{ft, in, m, mile, ...}\} \quad 4.1$$

where the dots in Set 4.1 represent all other length units not listed. For the dimension of absolute temperature, T, the set

$$T = \{\text{K, } ^\circ\text{R}\} \quad 4.2$$

is correct but the Set 4.3 is not,

$$T = \{\text{ } ^\circ\text{C, } ^\circ\text{F}\} \quad 4.3$$

because there is no constant K to express $^\circ\text{F} = \text{K} \cdot ^\circ\text{C}$. The relation between these quantities is given in Part I. In the case of temperature difference, however, the dimension of temperature may be defined by

$$T = \{\text{ } ^\circ\text{C, } ^\circ\text{F}\} \quad 4.4$$

In the list of the secondary quantities, can be included the list of the primary quantities along with any derived quantities such as energy, power, specific heat, thermal conductivity, heat transfer coefficient, viscosity, etc. The dimensions of secondary quantities can be had from their definitions in terms of the primary quantities. The dimensions of heat/energy, power, pressure, viscosity etc. are given in Table 1.1.

The ultimate objective of dimensional analysis is to combine many physical quantities/variables into a smaller number of dimensionless groups. The groups themselves appear in the final equation instead of many individual physical quantities. It should be made clear that dimensional analysis does not produce a numerical equation that can directly be used without any experimentation. It is rather a requirement to conduct experiments with the variables considered in the dimensional analysis and thereafter estimate the values of the exponents/coefficients of the equation developed by dimensional analysis using the test data and complete the equation.

It is important to note that even though dimensional analysis is a powerful tool in many engineering areas, it also suffers some drawbacks. From the dimensional analysis, no insight is gained about the mechanism of the process, the result may be meaningless if appropriate secondary quantities are not chosen, there is no way to know which group(s) is/are dominant and

which ones are not significant after the final expression is obtained, and, finally, the values of the dimensionless groups at which the process critically changes, are not estimated by this method.

Methods of Dimensional Analysis

The primary requirement in doing dimensional analysis of a physical process is to know about the process sufficiently. The selection of primary and secondary quantities depends largely on this knowledge. Also it helps if one knows what physical law(s) would be involved in the solution. The application of the principle of dimensional homogeneity in the methods of dimensional analysis is made in two ways.

Method 1

The following steps will describe this method in a way that it will be easier for a reader to follow them and solve a relevant problem.

- Step 1. List all the primary quantities and then the secondary quantities with appropriate symbols. This step is the most critical one. The success of the method depends on it. Here needed are the knowledge about the referred process, discretion, physical intuition, and experience in the method.
- Step 2. Write the secondary quantities in terms of the primary ones. This requires the knowledge in units and dimensions described in Part I.
- Step 3. Write a general functional equation for the process, using symbols, in terms of the secondary quantities decided in step 2, and assign arbitrary exponents to each of the independent secondary quantities in the right hand side of the functional equation. The dependent secondary quantity on the left hand side of the equation should have an exponent of unity.
- Step 4. Substitute the appropriate dimensional expressions from step 2.
- Step 5. Write simultaneous algebraic equations equating exponents of like dimensions on either side of the functional equation.
- Step 6. Select the principal exponents. This step says that the secondary quantities without having these principal exponents are repeated in the final equation. This step is extremely important, and it also requires all the experience, intuition, etc., about the process. A trial-and-error procedure may also be applied to get the desired equation in the end. The number of these principal exponents will decide the number of independent dimensionless groups on the right hand side of the final equation to be obtained in step 8.
- Step 7. Solve the equations obtained in step 5 for the rest of the exponents in terms of the principal ones chosen in step 6.

Step 8. Express the general functional equation in step 3 with the secondary exponents, and group the dimensions with the same exponent. Each of these groups should be dimensionless. Finally write the equation as a function of these groups instead of the secondary quantities.

Let us consider an example of heat transfer into a pipe. The rate of heat transfer per unit area, $[q/A]$ in W/m^2 is considered as the dependent variable. The choice of independent variables depends on the knowledge of the physical process, which in this example is the process of heat transfer.

From the characteristics of the process, it can be expected that the rate of heat transfer per unit area depends on (i) inside diameter of pipe, D , in m, (ii) temperature difference between the wall and the fluid, ΔT , in $^{\circ}C$ and fluid properties such as (iii) average velocity inside the pipe, V , in m/s, (iv) density, ρ , in kg/m^3 , (v) viscosity, μ , in Pa.s, (vi) specific heat, C_p , in $J/kg^{\circ}C$, and (vii) thermal conductivity, k , in $W/m.^{\circ}C$. It means that any change in any one of these seven independent secondary quantities will effect a change in the dependent secondary quantity, i.e. heat transfer rate. The empirical method of developing an equation relating all these variables requires that the effect of each be studied and determined while the rest are kept unchanged. It is a tedious and time-consuming process and, most often, the results thus obtained cannot be properly organized to get a useful relationship. On the other hand, a pure mathematical development of an equation correlating all these seven variables, seems a rather distant probability. We will now see how the dimensional approach simplifies the development of the governing equation.

Example 4.1. Develop an expression to estimate the rate of heat transfer per unit area across a heat exchanger pipe to the liquid inside, which is being heated by the hot wall. Assume that the wall temperature is greater than that of the liquid by a constant amount. Neglect the conversion of mechanical energy into heat by friction in comparison with the heat transferred to the liquid through the wall. Assume turbulent flow.

Solution. The general functional equation of this problem then can be written as

$$\frac{q}{A} = f(D, V, \rho, \mu, C_p, k, \Delta T) \quad 4.1.1$$

Now, Eq. (4.1.1) should have the same dimensions on its both the sides. The dimensional formula then will be

$$\frac{q}{A} = [D]^a [V]^b [\rho]^c [\mu]^d [C_p]^e [k]^f [\Delta T]^g \quad 4.1.2$$

When Eq. (4.1.2) is examined for the fundamental dimensions of each of its terms, it is seen that there are only four of them — mass, length, time, and temperature against seven unknown exponents, namely, letters a through g. In order to solve for the exponents in terms of two principal ones, there should be five primary quantities instead of four. Thus heat, [H], is added to the list of the primary quantities. In some other occasions if the need arises, force, [F], also can be considered as one of the primary quantities.

Dimensions of each term, adding heat, [H], to the list of primary quantities, are as follows:

$$\begin{aligned}\left[\frac{q}{A}\right] &= HL^{-2}T^{-1} \\ [D] &= L \\ [V] &= LT^{-1} \\ [\rho] &= ML^{-3} \\ [\mu] &= ML^{-1}T^{-1} \\ [C_p] &= HM^{-1}\theta^{-1} \\ [k] &= HL^{-1}T^{-1}\theta^{-1} \\ [\Delta T] &= \theta\end{aligned}$$

Now, substituting these dimensions in Eq. (4.1.2),

$$HL^{-2}T^{-1} = [L]^a[LT^{-1}]^b[ML^{-3}]^c[ML^{-1}T^{-1}]^d[HM^{-1}\theta^{-1}]^e[HL^{-1}T^{-1}\theta^{-1}]^f[\theta]^g$$

or

$$H^1M^0L^{-2}T^{-1}\theta^0 = H^{e+f}.M^{c+d+e}.L^{a+b-3c-d-f}.T^{-b-d-f}.\theta^{-e-f+g} \quad 4.1.3$$

Now comparing the exponents of the like dimensions/primary quantities on the left hand side and the right hand side of Eq. (4.1.3),

Exponents of H

$$1 = e + f \quad 4.1.4$$

Exponents of M

$$0 = c + d - e \quad 4.1.5$$

Exponents of L

$$-2 = a + b - 3c - d - f \quad 4.1.6$$

Exponents of T

$$-1 = -b - d - f \quad 4.1.7$$

Exponents of θ

$$0 = -e - f + g \quad 4.1.8$$

As we see, there are five equations and seven unknowns. Thus each exponent can be expressed in terms of any two of the other exponents. At this point, it is critical to choose the two principal exponents in terms of which the others should be expressed. It requires a thorough knowledge of the process in concern and a gathered experience in the art of dimensional analysis. For this particular problem of heat transfer, the principal secondary quantities are velocity and specific heat, meaning that exponents a, c, d, f, and g should be expressed in terms of exponents of the velocity and the specific heat, which are b and e, respectively.

From Eq. (4.1.4),

$$f = 1 - e \quad 4.1.9$$

From Eq. (4.1.5),

$$c + d = e \quad 4.1.10$$

From Eq. (4.1.7),

$$d + f = 1 - b \quad 4.1.11$$

From Eq. (4.1.8),

$$g - f = e \quad 4.1.12$$

Now, from Eqs. (4.1.9) and (4.1.11),

$$\begin{aligned} d + 1 - e &= 1 - b \\ d &= e - b \end{aligned} \quad 4.1.13$$

From Eqs. (4.1.10) and (4.1.13),

$$\begin{aligned} c + e - b &= e \\ c &= b \end{aligned} \quad 4.1.14$$

From Eqs. (4.1.9) and (4.1.12),

$$\begin{aligned} g - (1 - e) &= e \\ g &= 1 \end{aligned} \quad 4.1.15$$

Substituting values of c, d, and f from Eqs. (4.1.14), (4.1.13), and (4.1.9), respectively, into Eq. (4.1.6),

$$\begin{aligned} -2 &= a + b - 3b - (e - b) - (1 - e) \\ a &= b - 1 \end{aligned} \quad 4.1.16$$

Putting values of the exponents a, c, d, f, and g in terms of b and e in Eq. (4.1.2) yields

$$\frac{q}{A} = D^{b-1} \cdot V^b \cdot \rho^b \cdot \mu^{e-b} \cdot C_p^e \cdot k^{1-e} \cdot \Delta T$$

Now, separating quantities with exponents b and e,

$$\frac{q}{A} = (D^b \cdot V^b \cdot \rho^b \cdot \mu^{-b}) (C_p^e \cdot k^{-e} \cdot \mu^e) \cdot D^{-1} \cdot k \cdot \Delta T$$

$$\text{or } \frac{q \cdot D}{k \cdot A \cdot \Delta T} = \left[\frac{\rho \cdot V \cdot D}{\mu} \right]^b \left[\frac{C_p \cdot \mu}{k} \right]^e \quad 4.1.17$$

The expressions both on the left hand and the right hand side of Eq. (4.1.17) are dimensionless, and thus the equation is homogeneous. The right hand side of Eq. (4.1.17) can be expressed as a function of two dimensionless quantities $[\rho \cdot V \cdot D / \mu]$ and $[C_p \cdot \mu / k]$, which are known as the **Reynolds number**, N_{Re} , and **Prandtl number**, N_{Pr} , respectively.

So, Eq. (4.1.17) becomes

$$\frac{q \cdot D}{k \cdot A \cdot \Delta T} = C \cdot f(N_{Re}, N_{Pr}) \quad 4.1.18$$

The right hand side of Eq. (4.1.18) usually is multiplied by a constant, C . This is almost always done, and we will add this constant C to all the final expressions of the solutions of the problems obtained by dimensional analysis given in this text. The form of the above function should be developed with experimental data correlating the groups on the right hand side with the group on the left hand side of Eq. (4.1.18). It is thus seen that experimental values can be used to correlate only three groups of variables in Eq. (4.1.18), which is much easier than correlating all the variables of Eq. (4.1.1) that we started with.

If instead of exponents b and e , any other two were chosen as independent exponents, again three dimensionless groups would have been obtained. For example, if c and f are chosen as the independent ones, we will end up with

$$\frac{q \cdot D}{A \cdot C_p \cdot \mu \cdot \Delta T} = [N_{Re}]^c [N_{Pr}]^f$$

$$\text{or } \frac{q \cdot D}{A \cdot C_p \cdot \mu \cdot \Delta T} = C \cdot f_1(N_{Re}, N_{Pr}) \quad 4.1.19$$

Similarly, if b and f were selected, the end result would be

$$\frac{q}{A \cdot V \cdot \rho \cdot C_p \cdot \Delta T} = C \cdot f_2(N_{Re}, N_{Pr}) \quad 4.1.20$$

In this way various other combinations can be found. The easier way to obtain such myriad dimensionless groups is by multiplying and dividing the left hand side and right hand side of any of the expressions first obtained with reciprocals or multiples of the dimensionless groups on the right hand side of

the equation (N_{Re} and/or N_{Pr} in this case). If Eq. (4.1.17) is divided on both sides by N_{Pr} ($= C_p \mu / k$), Eq. (4.1.19) is obtained. Similarly, Eq. (4.1.17) divided on both sides by ($N_{Re} \cdot N_{Pr}$), yields Eq. (4.1.20).

An expression for the heat transfer coefficient, h_i at the interface of the cylinder and the fluid inside can be had from the following equation :

$$q = A \cdot h_i \cdot \Delta T$$

$$\text{or } h_i = \frac{q}{A \cdot \Delta T} \quad 4.1.21$$

From Eqs. (4.1.18) and (4.1.21),

$$\frac{h_i \cdot D}{k} = f(N_{Re}, N_{Pr})$$

where the dimensionless group $[h_i \cdot D / k]$ is called the **Nusselt number**, N_{Nu} .

Thus

$$N_{Nu} = C \cdot f(N_{Re}, N_{Pr}) \quad 4.1.22$$

A recognized empirical correlation for sharp-edged entrances, using Eq. (4.1.22) is

$$N_{Nu} = 0.023 \left[1 + \left(\frac{D}{L} \right)^{0.7} \right] N_{Re}^{0.8} \cdot N_{Pr}^{1/3} \left(\frac{\mu}{\mu_w} \right)^{0.14} \quad 4.1.23$$

where μ_w is the absolute viscosity at the wall temperature.

Let us solve two more problems, one in fluid flow and the other in mass transfer by the Method 1.

Example 4.2. Develop an expression for pressure drop due to Newtonian fluid flow inside a smooth, long, circular and straight tube for laminar or turbulent condition.

Solution. From the knowledge of the process, the pressure drop, Δp , in Pa, should be dependent on the characteristics of the tube such as its (i) length, L , in m and (ii) diameter, D , in m, and fluid properties such as its (iii) velocity, V , in m/s, (iv) density, ρ , in kg/m^3 , and (v) viscosity, μ , in Pa.s.

The theoretical equation then is

$$\Delta p = f(L, D, V, \rho, \mu) \quad 4.2.1$$

The dimensional formula for Eq. (4.2.1) will be

$$\Delta p = [L]^a [D]^b [V]^c [\rho]^d [\mu]^e \quad 4.2.2$$

Here it is not necessary to add heat or force as the primary quantity because there are three basic quantities against five exponents in this problem, which finally are adequate to obtain an equation with two independent dimensionless groups.

Dimensions of each secondary quantity in Eq. (4.2.2) are

$$\begin{aligned} [p] &= ML^{-1}T^{-2} \\ [L] &= L \\ [D] &= L \\ [V] &= LT^{-1} \\ [\rho] &= ML^{-3} \\ [\mu] &= ML^{-1}T^{-1} \end{aligned}$$

Substituting these into Eq. (4.2.2),

$$\begin{aligned} ML^{-1}T^{-2} &= [L]^a [L]^b [LT^{-1}]^c [ML^{-3}]^d [ML^{-1}T^{-1}]^e \\ \text{or } ML^{-1}T^{-2} &= M^{d+e} L^{a+b+c-3d-e} T^{-c-e} \end{aligned} \quad 4.2.3$$

Equating exponents of each primary quantity between the left hand and the right hand side of Eq. (4.2.3),

Exponents of M

$$1 = d + e \quad 4.2.4$$

Exponents of L

$$-1 = a + b + c - 3d - e \quad 4.2.5$$

Exponents of T

$$-2 = -c - e \quad 4.2.6$$

Now, considering a and e as the two independent exponents, the rest (b , c , and d) can be expressed in terms of these two in the following manner :

From Eq. (4.2.4),

$$d = 1 - e \quad 4.2.7$$

From Eq. (4.2.6),

$$c = 2 - e \quad 4.2.8$$

In Eq. (4.2.5), substituting the values of c and d from Eqs. (4.2.8) and (4.2.7), respectively,

$$\begin{aligned} b &= -1 - a - c + 3d + e \\ &= -1 - a - (2 - e) + 3(1 - e) + e \\ &= -a - e \end{aligned} \quad 4.2.9$$

Now, Eq. (4.2.2), when used with Eqs. (4.2.7) through (4.2.9), gives

$$\Delta p = L^a \cdot D^{-a-e} \cdot V^{-e+2} \cdot \rho^{1-e} \cdot \mu^e$$

$$\text{or } \left[\frac{\Delta p}{\rho \cdot V^2} \right] = \left[\frac{L}{D} \right]^a \left[\frac{\mu}{\rho \cdot V \cdot D} \right]^e$$

$$= \left[\frac{L}{D} \right]^a \left[\frac{1}{N_{Re}} \right]^e \quad 4.2.10$$

$$\left[\frac{\Delta p}{\rho \cdot V^2} \right] = C \cdot f \left(\frac{L}{D}, N_{Re} \right) \quad 4.2.11$$

The Fanning friction factor, f , in the case of the flow of fluid inside a conduit, is given in the literature as

$$f = \frac{-\Delta p \cdot D}{2L \cdot \rho \cdot V^2} \quad 4.2.12$$

For the laminar flow of a Newtonian fluid

$$f = \frac{16}{N_{Re}} \quad 4.2.13$$

Eq. (4.2.12) is combined with Eq. (4.2.13), to get

$$\frac{-\Delta p}{\rho \cdot V^2} = \frac{32 \cdot \left[\frac{L}{D} \right]}{N_{Re}} \quad 4.2.14$$

When Eqs. (4.2.10) and (4.2.14) are compared with each other, it becomes obvious that the latter has been developed from the former through experimentation. Equation (4.2.11) can equally be used for laminar as well as turbulent conditions. Only the experiments are to be designed differently in either case to determine the exponents a and e .

Example 4.3. Develop an expression to calculate the coefficient of mass transfer between fluids or between fluids and solids when the area of mass transfer is known.

Solution. Individual mass transfer coefficient, k_m , is a measure of the resistance to mass transfer by molecular and turbulent diffusion, independent of any drift of the entire phase. From the mechanism of mass transfer, it can

be expected that the coefficient, k_m , (in $\text{kg.mol/m}^2\text{.s.mole fraction}$) will depend on (i) diffusivity, α_m in m^2/s and on the variables that control the character of fluid flow, e.g., (ii) mass velocity, G , in $\text{kg/m}^2\text{.s}$, (iii) viscosity, μ , in Pa.s and (iv) density, ρ , in kg/m^3 , and (v) linear dimension, D , in m . The shape of the interface does influence the process of mass transfer, and, as a result, a different relation should appear for each shape.

For any given shape, the mass transfer coefficient can be expressed as a function of all the five variables mentioned above, i.e.,

$$k_m = f(\alpha_m, D, G, \mu, \rho) \quad 4.3.1$$

and the dimensional formula as

$$k_m = [\alpha_m]^a [D]^b [G]^c [\mu]^d [\rho]^e \quad 4.3.2$$

Dimensions of each secondary quantity in Eq. (4.3.2) are

$$\begin{aligned} [k_m] &= \text{ML}^2\text{T}^{-1} \\ [\alpha_m] &= \text{L}^2\text{T}^{-1} \\ [D] &= \text{L} \\ [G] &= \text{ML}^{-2}\text{T}^{-1} \\ [\mu] &= \text{ML}^{-1}\text{T}^{-1} \\ [\rho] &= \text{ML}^{-3} \end{aligned}$$

These dimensions, substituting into Eq. (4.3.2), yield

$$\begin{aligned} \text{ML}^{-2}\text{T}^{-1} &= [\text{L}^2\text{T}^{-1}]^a [\text{L}]^b [\text{ML}^{-2}\text{T}^{-1}]^c [\text{ML}^{-1}\text{T}^{-1}]^d [\text{ML}^{-3}]^e \\ \text{or } \text{ML}^{-2}\text{T}^{-1} &= \text{M}^{c+d+e} \cdot \text{L}^{2a+b-2c-d-3e} \cdot \text{T}^{-a-c-d} \end{aligned} \quad 4.3.3$$

Now, equating the exponents of each of the primary quantities on both the sides of Eq. (4.3.3),

Exponents of M

$$1 = c + d + e \quad 4.3.4$$

Exponents of L

$$-2 = 2a + b - 2c - d - 3e \quad 4.3.5$$

Exponents of T

$$-1 = -a - c - d \quad 4.3.6$$

In this problem, the principal secondary quantities chosen are diameter and density, which implies that exponents a , c , and d should be expressed in terms of exponents of diameter and density.

From Eq. (4.3.4),

$$c + d = 1 - e \quad 4.3.7$$

From Eq. (4.3.6),

$$c + d = 1 - a \quad 4.3.8$$

From Eqs. (4.3.7) and (4.3.8),

$$a = e \quad 4.3.9$$

From Eqs. (4.3.5) and (4.3.9),

$$\begin{aligned} -2 &= 2e + b - 2c - d - 3e \\ \text{or } 2c + d &= 2 + b - e \end{aligned} \quad 4.3.10$$

Now, using Eqs. (4.3.7) and (4.3.10), c is obtained as

$$c = 1 + b \quad 4.3.11$$

From Eqs. (4.3.4) and (4.3.11),

$$\begin{aligned} d &= 1 - e - (1 + b) \\ &= -b - e \end{aligned} \quad 4.3.12$$

Substituting the values of a, c, and d from Eqs. (4.3.9), (4.3.11), and (4.3.12), respectively, into Eq. (4.3.2),

$$\begin{aligned} k_m &= [\alpha_m]^e [D]^b [G]^{1+b} [\mu]^{-b-e} [\rho]^e \\ \text{or } \left[\frac{k_m}{G} \right] &= \left[\frac{D \cdot G}{\mu} \right]^b \left[\frac{\alpha_m \cdot \rho}{\mu} \right]^e \end{aligned} \quad 4.3.13$$

The left hand side of Eq. (4.3.13) can be multiplied by $\overline{M}$, the average molecular weight of the entire phase to make it more consistent and specific. The dimensionless group $[\mu/(\alpha_m \cdot \rho)]$ is called the **Schmidt number**, N_{Sc} . Now, multiplying Eq. (4.3.13) by $(N_{Re} \cdot N_{Sc})$, we get

$$\frac{k_m \cdot D \cdot \overline{M}}{\rho \cdot \alpha_m} = f(N_{Re}, N_{Sc})$$

The dimensionless group on the left hand side of the above equation is called the **Sherwood number**, N_{Sh} , and it can be written as

$$N_{Sh} = C.f(N_{Re}, N_{Sc}) \quad 4.3.14$$

For the flow past single spheres, the following mass transfer equation is reported in the literature:

$$N_{Sh} = 2 + 0.6 N_{Re}^{0.5} . N_{Sc}^{1/3} \quad 4.3.15$$

To obtain Eq. (4.3.15), obviously Eq. (4.3.14) was used, and the coefficients were obtained conducting experiments.

There is a second method of dimensional analysis that uses the famous pi theorem of Buckingham. Both the methods have some basic concepts in common. Before explaining Method 2, the pi theorem is described.

Pi Theorem of Buckingham

In any physical or chemical process, the dependent variable is controlled by several other variables. Any change in these independent variables causes a change in the dependent one when the rest of the independent variables remain unchanged without being influenced. This requirement can mathematically be expressed as

$$Y = f(X_1, X_2, X_3, \dots, X_n) \quad 4.5$$

where Y is the dependent variable and $X_1, X_2, X_3, \dots, X_n$ are the independent ones. The dependent variable is written as a function of n independent variables.

The relation in Eq. (4.5) can be expressed as a series of a number of terms each composed of the product of the independent variables raised to a suitable power. Thus

$$Y = X_1^{a_1} . X_2^{b_1} . X_3^{c_1} \dots X_n^{r_1} + X_1^{a_2} . X_2^{b_2} . X_3^{c_2} \dots X_n^{r_2} + \dots \quad 4.6$$

where the exponents $a_1, b_1, c_1, \dots, r_1$ and $a_2, b_2, c_2, \dots, r_2$ are numbers only without dimensions. If Eq. (4.6) is now divided on both sides by the first term on the right hand side, i.e., $(X_1^{a_1} . X_2^{b_1} . X_3^{c_1} \dots X_n^{r_1})$, we obtain

$$\frac{Y}{X_1^{a_1} . X_2^{b_1} . X_3^{c_1} \dots X_n^{r_1}} = 1 + X_1^{(a_2-a_1)} . X_2^{(b_2-b_1)} . X_3^{(c_2-c_1)} \dots X_n^{(r_2-r_1)} \quad 4.7$$

The first term on the right hand side of Eq. (4.7) is unity, i.e., dimensionless. Now, the dimensional homogeneity requires that all the rest of

the terms of Eq. (4.7) must also be dimensionless. The dimensionless group on the left hand side of Eq. (4.7) is denoted by π_1 . Other dimensionless groups $\pi_2, \pi_3, \pi_4, \dots, \pi_{n+1-m}$ can be formed by dividing Eq. (4.6) successively by each of the groups on the right hand side of the same equation. The quantities $(n+1)$ and m are the total number of variables (dependent and independent secondary quantities) and the number of physical dimensions (primary quantities) involved, respectively. The total number of π terms is $(n+1-m)$ and thus will be a function of the primary quantities, m and total number of secondary quantities involved in the process. Each π term can be expressed as the function of the rest of the π terms, such as

$$\pi_1 = C.f_1(\pi_2, \pi_3, \dots, \pi_{n+1-m}) \quad 4.8$$

where f_1 is used to denote another function of the indicated dimensionless groups, and C is a constant. As it is observed, the number of groups to be formed by the general functional equation should be $(n+1-m)$.

Method 2

The following steps will describe this method in a way that it will be easier for the reader to follow them and solve a relevant problem later. Some of the steps here are the same as those of Method 1, but, for the convenience of the reader, all of them are discussed here.

- Step 1. List all the primary quantities and then the secondary quantities with appropriate symbols. This step is the most critical one. The success of the method depends on it. Needed here are knowledge about the referred process, discretion, physical intuition, and experience in the method.
- Step 2. Write the general functional equation with all the variables selected in step 1. This will include the dependent secondary quantity as a function of the independent secondary quantities.
- Step 3. Write the secondary quantities in terms of the primary ones. This requires the knowledge in units and dimensions described in Part I.
- Step 4. Select the principal secondary quantities. This step says that these secondary quantities are repeated in the final equation. This step is extremely important and also requires all the experience, intuition etc., about the process. A trial-and-error procedure may also be applied to get the desired equation in the end.
- Step 5. Write the dimensionless equations for π terms substituting the appropriate dimensional expressions from step 2 and assigning arbitrary exponents to each of the secondary quantities. The number of such equations will depend on the total number of secondary quantities and the number of primary quantities selected in step 1.

The present method requires that the total number of secondary quantities be equal to four or more. The first pi term should contain the dependent secondary quantity with its exponent unity. Each of the rest of the pi terms should contain only one of the independent secondary quantities with the exponent unity, which were not considered with principal exponents in step 3.

- Step 6. Write simultaneous algebraic equations equating exponents of like dimensions on either side of the equations for pi.
- Step 7. Solve each of the equations obtained in step 5 for zero, because pi terms are dimensionless and calculate the values of the exponents.
- Step 8. Express each equation of pi as a dimensionless group using the values of the exponents calculated in step 7 and finally write the complete equation as a function of these groups instead of the secondary quantities.

Let us solve the following problem very similar to Problem 4.2 by the Buckingham's pi theorem. The present example differs from the previous one by the fact that here the tube is considered as rough, introducing a secondary quantity, roughness, e , in m in it.

Example 4.4. An incompressible fluid flows in a long, circular, straight and rough tube. Derive a relationship between the variables involved in this process to express the pressure loss in the tube due to fluid flow.

Solution. The primary quantities selected are (1) length, L , (2) mass, M , and (3) time, T . The dependent secondary quantity is pressure loss, Δp , in Pa and the independent secondary quantities chosen are (1) diameter of tube, D , in m, (2) length of tube, L , in m, (3) roughness of tube, e , in m, (4) absolute viscosity of fluid, μ , in Pa.s, (5) velocity of fluid, V , in m/s, (6) density, ρ in kg/m^3 , and (7) acceleration due to gravity, g , in m/s^2 .

The general functional equation is then

$$\Delta p = f(D, \mu, V, \rho, L, e, g) \quad 4.4.1$$

In dimensional forms, the secondary quantities will be,

$$\begin{aligned} [\Delta p] &= ML^{-1}T^{-2} \\ [D] &= L \\ [\mu] &= ML^{-1}T^{-1} \\ [V] &= LT^{-1} \\ [\rho] &= ML^{-3} \\ [L] &= L \\ [e] &= L \\ [g] &= LT^{-2} \end{aligned}$$

At this point three principal secondary quantities are selected — diameter, velocity, and density. The total number of independent secondary quantities is seven ($n=7$) in this problem, and the number of primary quantities selected is three ($m=3$). So, the total number of dimensionless pi terms will be $(n+1-m)$, or five. Then the first equation of pi according to step 5, is

$$\pi_1 = [L]^a [LT^{-1}]^b [ML^{-3}]^c [ML^{-1}T^{-2}] \quad 4.4.2$$

$$\text{or } L^0 M^0 T^0 = L^{a+b-3c-1} M^{c+1} T^{-b-2}. \quad 4.4.3$$

Equating the exponents of L, M, and T separately of both the sides of Eq. (4.4.3),

Exponents of L

$$0 = a + b - 3c - 1 \quad 4.4.4$$

Exponents of M

$$0 = c + 1 \quad 4.4.5$$

Exponents of T

$$0 = -b - 2 \quad 4.4.6$$

Solving Eqs. (4.4.4) through (4.4.6),

$$a = 0; b = -2 \text{ and } c = -1$$

Substituting these values of a, b, and c into Eq. (4.4.2),

$$\pi_1 = \frac{\Delta p}{\rho \cdot V^2} \quad : \text{ Euler number, } N_{Eu}$$

The second equation of pi is

$$\pi_2 = [L]^a [LT^{-1}]^b [ML^{-3}]^c [ML^{-1}T^{-1}] \quad 4.4.7$$

The values of exponents a, b, and c are calculated the same way as above to give

$$a = -1; b = -1 \text{ and } c = -1$$

Thus,

$$\pi_2 = \frac{\mu}{\rho \cdot V \cdot D} \quad : \frac{1}{\text{Reynolds number}}$$

The third equation of pi is

$$\pi_3 = [L]^a [LT^{-1}]^b [ML^{-3}]^c [L] \quad 4.4.8$$

Here, $a = -1$, $b = 0$, and $c = 0$, thus

$$\pi_3 = \frac{L}{D}$$

The fourth equation of π is the same as Eq. (4.4.8) because the independent secondary quantity in this case with exponent of unity is the roughness, e , which has the same dimension $[L]$ as the previous secondary quantity length of the tube, L . The values of a , b , and c should then be

$$a = -1, b = 0, \text{ and } c = 0 \text{ and}$$

$$\pi_4 = \frac{e}{D} \quad : \text{Roughness factor.}$$

Finally the fifth and the last equation of π is

$$\pi_5 = [L]^a [LT^{-1}]^b [ML^{-3}]^c [LT^{-2}] \quad 4.4.9$$

Solving for a , b and c gives

$$a = 1; b = -2, \text{ and } c = 0$$

$$\pi_5 = \frac{g \cdot D}{V^2} \quad : \frac{1}{\text{Froude number}} = \frac{1}{N_{Fr}}$$

Now, putting all the π terms in Eq. (4.8), the final equation becomes

$$N_{Eu} = C \cdot f_1(N_{Re}, \frac{L}{D}, \frac{e}{D}, N_{Fr}) \quad 4.4.10$$

Example 4.5. A spherical body is fully submerged in an incompressible fluid and is moving with a constant velocity. Derive an expression for the drag force on the body as a function of the variables involved in the process.

Solution. Based on our knowledge of this process, we shall assume that the drag force, F , in N is a function of (1) diameter of the body, D , in m , (2) velocity of the body, V , in m/s , (3) density of fluid, ρ , in kg/m^3 and (4) viscosity of fluid, μ in $Pa \cdot s$. These are the secondary quantities. The primary quantities are (1) mass, M , (2) length, L , and (3) time, T .

The general functional equation for drag force can be written as

$$F = f(D, V, \rho, \mu) \quad 4.5.1$$

The dimensional representations of the secondary quantities in terms of the primary ones are

$$\begin{aligned}
[F] &= MLT^{-2} \\
[D] &= L \\
[V] &= LT^{-1} \\
[\rho] &= ML^{-3} \\
[\mu] &= ML^{-1}T^{-1}
\end{aligned}$$

The principal secondary quantities selected at this point are diameter, velocity, and density. Thus the number of dimensionless groups (pi terms) in the solution should be equal to two (4+1-3).

The first pi term is

$$\pi_1 = [L]^a [LT^{-1}]^b [ML^{-3}]^c [MLT^{-2}] \quad 4.5.2$$

Now, equating the exponents of M, L, and T to zero,

$$a = -2, b = -2, \text{ and } c = -1$$

Substituting these values into Eq. (4.5.2),

$$\pi_1 = \frac{F}{D^2 \cdot V^2 \cdot \rho}$$

The second pi term is

$$\pi_2 = [L]^a [LT^{-1}]^b [ML^{-3}]^c [ML^{-1}T^{-1}] \quad 4.5.3$$

Equating the exponents of M, L, and T to zero,

$$a = -1, b = -1, \text{ and } c = -1$$

Substituting these values in Eq. (4.5.3),

$$\pi_2 = \frac{\mu}{D \cdot V \cdot \rho}$$

Now, substituting the expressions for the pi terms into Eq. (4.8) the final equation becomes

$$\frac{F}{D^2 \cdot V^2 \cdot \rho} = C \cdot f_1(N_{Re}) \quad 4.5.4$$

Seven dimensionless groups have been developed in the course of solving problems given in Examples 4.1 through 4.5. It is advantageous to have an exhaustive list of such dimensionless groups that one may see when developing mathematical models by the dimensional analysis technique. In

Table 4.1 a list of such numbers in the areas of fluid mechanics, mass transfer, and heat transfer has been provided for ready reference.

Table 4.1. List of dimensionless groups

Name of group	Symbol	Expression
Biot number	N_{Bi}	$\frac{h \cdot l}{k}$
Euler number	N_{Eu}	$\frac{\Delta p}{\rho \cdot V^2}$
Fourier number	N_{Fo}	$\frac{\alpha \cdot T}{l^2}$
Froude number	N_{Fr}	$\frac{V^2}{g \cdot L}$
Grashof number	N_{Gr}	$\frac{L^3 \cdot \rho^2 \cdot \beta \cdot g \cdot \Delta T}{\mu^2}$
Graetz number	N_{Gz}	$\frac{m \cdot C_p}{k \cdot L}$
Lewis number	N_{Le}	$\frac{h}{k_m \cdot C_p}$
Mach number	N_{Ma}	$\frac{V}{a_c}$
Nusselt number	N_{Nu}	$\frac{h \cdot l}{k}$
Peclet number	N_{Pe}	$\frac{D \cdot V}{\alpha}$
Prandtl number	N_{Pr}	$\frac{C_p \cdot \mu}{k}$
Reynolds number	N_{Re}	$\frac{\rho \cdot V \cdot D}{\mu}$
Schmidt number	N_{Sc}	$\frac{\mu}{\rho \cdot \alpha_v}$
Sherwood number	N_{Sh}	$\frac{k_m \cdot D \cdot \bar{M}}{\rho \cdot \alpha_m}$
Stanton number	N_{St}	$\frac{h}{C_p \cdot \rho \cdot V}$
Weber number	N_{We}	$\frac{D \cdot \rho \cdot V^2}{\sigma}$

Model Testing for Scaling Up

It is much more convenient to test the model of an equipment in the laboratory and be able to determine the performance of a full-scale equipment on a completely theoretical basis than conducting time consuming experiments on the real equipment. A simplified mathematical analysis combined with experimental data on a scale model is capable of estimating the performance of full-sized equipment. The application of dimensional analysis to such a problem allows us to use the derived dimensionless groups in a systematic manner to estimate the performance on any other scale.

The full-scale equipment is called the prototype. In order to be able to apply the results obtained by model testing in scaling up the real equipment, it is necessary that there be similarity between the model and the prototype. This similarity implies that the shape, the motion, and the force must each have a constant but not necessarily the same ratio between the model and the prototype. These three forms of similarity are known as (1) geometric similarity associated with shape, (2) kinematic similarity associated with motion, and (3) dynamic similarity connected with force.

Geometric similarity

For this similarity, it is necessary for the model to be an exact scale replica of the prototype. The dimensionless groups such as length to diameter ratio (L/D), and relative roughness (e/D) in the illustrative Problem 4.4 are used to determine the dimensions of the prototype, e.g., its length, diameter, and surface roughness.

In the case of geometric similarity,

$$\left[\frac{L}{D} \right]_m = \left[\frac{L}{D} \right]_p \quad 4.9$$

$$\text{and } \left[\frac{e}{D} \right]_m = \left[\frac{e}{D} \right]_p \quad 4.10$$

where the subscripts m and p denote model and prototype, respectively.

Thus

$$\frac{L_m}{L_p} = \frac{D_m}{D_p} = \frac{e_m}{e_p} \quad 4.11$$

Now, if any one ratio is known, the other ratios can be used to calculate the parameters of the prototype. For example, if the diameter of the prototype D_p is decided,

$$L_p = L_m \left[\frac{D_p}{D_m} \right] \quad 4.12$$

$$\text{and } e_p = e_m \left[\frac{D_p}{D_m} \right] \quad 4.13$$

Kinematic similarity

The requirement for kinematic similarity is that the motion of the model and the prototype as well as the motion of the fluid particle in either system be similar. If we define the geometric similarity factor, velocity similarity factor, acceleration similarity factor, and time scale similarity factor as K_g , K_v , K_a , and K_t , respectively the condition for kinematic similarity will be

$$K_g = K_v \cdot K_t \quad 4.14$$

$$\text{and } K_g = K_a \cdot K_t^2 \quad 4.15$$

where

$$K_g = \frac{L_p}{L_m} \quad K_v = \frac{V_p}{V_m} \quad K_a = \frac{a_p}{a_m}, \text{ and } K_t = \frac{t_p}{t_m}$$

Examining Eqs. (4.14) and (4.15), it can be concluded that, if there is a tenfold scaling up, and it is decided that the prototype will have the same velocity as the model, the time scale for the prototype should also be tenfold greater than that of the model. On the other hand, if the time scales are kept the same, the velocity of the prototype should be ten times greater than that of the model. The acceleration of the prototype will go by the scale factor alone if the time factor is kept unchanged for both the model and the prototype. When the acceleration is not allowed to change, a tenfold scaling up will result in an increase of time scale by 10.

Dynamic similarity

It requires that the ratio of pressures at identical points on the model and the prototype be constant. Also the ratio of any forces at identical points on the model and the prototype must be the same if the first requirement is valid at the same points.

If the force similarity factor and mass similarity factor are denoted by K_f and K_m , respectively, the condition for dynamic similarity is given by

$$K_f = K_m \frac{K_g}{K_t^2} \quad 4.16$$

where

$$K_f = \frac{F_p}{F_m}, \text{ and } K_m = \frac{M_p}{M_m}$$

Shames (1962) stated that dimensional analysis will yield the dimensionless groups in a flow, for example, for which we must duplicate at least all but one in geometrically similar flows to achieve dynamic similarity. Let us consider Example 4.4 to illustrate this principle.

The four dimensionless groups on the right hand side of Eq. (4.4.10) are Reynolds number ($\rho \cdot V \cdot D / \mu$), Froude number ($V^2 / g \cdot D$), the roughness factor (e/D), and the length to diameter ratio (L/D). For the flow in a model system to be similar to that in a prototype, it is required that these four dimensionless groups be equal in both the model and the prototype.

If it is done, the dimensionless group on the left hand side of Eq. (4.4.10), which is the Euler number ($\Delta p / \rho \cdot V^2$), also will be the same in both the cases. This means that the model and the prototype are made in the same scale and that the flows are identical. This is meaningless, obviously defeating the very purpose of scaling up.

Fortunately in most cases one effect is usually dominant over the others, and it is possible to retain the dimensionless groups incorporating this dominant effect and correct the other dimensionless groups that are not equal in the model and the prototype.

In Example 4.4, the dominant dimensionless group is the Reynolds number, which expresses the ratio of inertia force to friction force. Also, for similarity, the necessary and sufficient condition is that the geometric ratio between the model and the prototype be constant. If the Reynolds number of both systems is equal, to meet the similarity, it would follow that the dependent secondary number in Eq. (4.4.10), which is ($\Delta p / \rho \cdot V^2$), must be equal for both the model and the prototype.

Therefore,

$$\left[\frac{\Delta p}{\rho \cdot V^2} \right]_m = \left[\frac{\Delta p}{\rho \cdot V^2} \right]_p \quad 4.17$$

Now, using Eq. (4.16), substituting the dimension of (Δp) which is $[FL^{-2}]$, into Eq. (4.17),

$$\begin{aligned} K_f &= \frac{F_p}{F_m} \\ &= \frac{[V_p^2 \cdot \rho_p \cdot L_p^2]}{[V_m^2 \cdot \rho_m \cdot L_m^2]} \end{aligned} \quad 4.18$$

Using the definitions of K_m , K_g , and K_t given earlier, Eq.(4.18) becomes

$$K_f = \frac{K_m \cdot K_g}{K_t^2}$$

which is the criterion for dynamic similarity given earlier by Eq. (4.16).

Now, let us illustrate the method of scaling up with the help of the following problem.

Example 4.6. Estimate the drag force on a prototype sphere fully submerged in water when it reaches its terminal velocity by testing a model sphere in air. The model is one tenth the size of the prototype, and the drag force on it is 10 N.

Solution. For similarity, the Reynolds number in the prototype and the model must be equal. Thus,

$$\left[\frac{\rho \cdot V \cdot D}{\mu} \right]_p = \left[\frac{\rho \cdot V \cdot D}{\mu} \right]_m \quad 4.6.1$$

From Eq. (4.6.1),

$$\frac{V_p}{V_m} = \left[\frac{\rho_m}{\rho_p} \right] \left[\frac{D_m}{D_p} \right] \left[\frac{\mu_p}{\mu_m} \right] \quad 4.6.2$$

Now, considering roughly that water is 1000 times denser than air, and its viscosity is 100 times the viscosity of air,

$$\frac{\rho_m}{\rho_p} = \frac{1}{100} \text{ and } \frac{\mu_p}{\mu_m} = 100$$

Since the model is one-tenth of the prototype,

$$\frac{D_m}{D_p} = \frac{1}{100}$$

Substituting the values of these ratios in Eq. (4.6.2),

$$\frac{V_p}{V_m} = \frac{1}{1000} \cdot \frac{1}{10} \cdot 100 = \frac{1}{100}$$

From the dimensional analysis of this problem, Eq. (4.5.4) was previously derived. For similarity, the dimensionless group on the left hand side of this equation must be equal for both the model and the prototype, that is

$$\left[\frac{F}{\rho \cdot D^2 \cdot V^2} \right]_m = \left[\frac{F}{\rho \cdot D^2 \cdot V^2} \right]_p \quad 4.6.3$$

From Eq. (4.6.3),

$$\frac{F_m}{F_p} = \left[\frac{\rho_m}{\rho_p} \right] \left[\frac{D_m}{D_p} \right]^2 \left[\frac{V_m}{V_p} \right]^2 = \left[\frac{1}{1000} \right] \left[\frac{1}{10} \right]^2 [100]^2 = 0.1$$

$$\text{So, } F_p = 10 \cdot F_m = 10(10) = 100 \text{ N}$$

Therefore, the prototype will experience a drag force of 100 N. It has been verified that the drag force on the prototype will continue to be ten times of that on the model, irrespective of their size ratios. The factors that affect it are the properties of the mediums in which they are submerged.

Symbols

A	Area, m ²
a, b, c, d, e, f, g	Exponents of dimensional formula
a _c	Acoustical velocity at pipe entrance, m/s
C	Constant in equations. of dimensional analysis
C _p	Specific heat, J/kg.°C
D	Diameter, m
e	Roughness, m
F	Force, N
f	Function of/Fanning friction factor
G	Mass velocity, kg/m ² .s
g	Acceleration due to gravity, m/s ²
H	Heat energy, J
h _i	Heat transfer coefficient, W/m ² .°C
K	Constant/Similarity factor
k	Thermal conductivity, W/m.°C
k _m	Mass transfer coefficient., kg.mol/m ² .mole fraction.
L	Length, m
l	Characteristic dimension, m
M	Mass, kg
$\bar{M}$	Average molecular weight
m	Mass flow rate, kg/s
N _{Bi}	Biot number

N_{Eu}	Euler number
N_{Fo}	Fourier number
N_{Fr}	Froude number
N_{Gr}	Grashof number
N_{Gz}	Graetz number
N_{Le}	Lewis number
N_{Ma}	Mach number
N_{Nu}	Nusselt number
N_{Pe}	Peclet number
N_{Pr}	Prandtl number
N_{Re}	Reynolds number
N_{Sc}	Schmidt number
N_{Sh}	Sherwood number
N_{St}	Stanton number
N_{We}	Weber number
p	Pressure, Pa
q	Heat flow rate, W
T	Time, s
V	Velocity of flow, m/s
α	Thermal diffusivity, m^2/s
α_m	Mass diffusivity, m^2/s
α_v	Volumetric diffusivity, m^2/s
β	Coefficient of expansion, 1/absolute temperature
Δ	Small amount of/Difference of
μ	Absolute viscosity, Pa.s
ρ	Density, kg/m^3
σ	Surface tension, N/m
θ	Temperature, $^{\circ}C$

Subscripts

a	Acceleration
f	Force
g	Geometry
m	Model/Mass
p	Prototype
v	Velocity
t	Time

References

1. **Bennett C. O. and Myers, J. E.**, *Momentum, Mass and Heat Transfer*. Third Edition. McGraw-Hill Book Co., New York, 1982.
2. **Giles, R. V., Evett, J.B. and C. Liu**, *Schaum's Outline of Theory and Problems of Fluid Mechanics and Hydraulics*. Third Edition. Schaum Publishing Co., New York, 1994.
3. **Granet, I.**, *Fluid Mechanics for Engineering Technology*. Third Edition, Prentice-Hall, Inc., NJ, 1989.
4. **Langhaar, H. L.**, *Dimensional Analysis and Theory of Models*. John Wiley & Sons, New York, 1951.
5. **McCabe, W. L. and Smith, J. C.**, *Unit Operations of Chemical Engineering*. McGraw-Hill Book Co., New York, 1976.
6. **Shames, I. H.**, *Mechanics of Fluids*. Third Edition, McGraw-Hill Book Co., New York, 1992.
7. **Streeter, V. L. and Wylie, E. B.**, *Fluid Mechanics*. Eighth Edition, McGraw-Hill Book Co., New York, 1985.
8. **Vennard, J. K. and Street, R. L.**, *Elementary Fluid Mechanics*. Sixth Edition. John Wiley & Sons, New York, 1982.

PART V

APPLICATIONS

So far a lot of theories have been discussed, and, in some cases, a few example problems are solved. To get the essence of all the theories, it is necessary to solve some real-world problems applying these theories. This part will be devoted to solving five such application problems very pertinent to food and agricultural engineering professions. The solutions will include semi-detailed treatments of the development of mathematical models/equations and related complete programming in FORTRAN. Programs will use the relevant subroutines developed in earlier chapters.

In food and agricultural engineering areas, air is a very important substance/medium that is used to cool or dry a biological product and an appropriate knowledge in air–water vapor properties is a necessity. Thus the first application problem has been chosen to solve for the properties of the air–vapor mixture when any two independent properties are known.

A finned tube heat exchanger is very useful where heat dissipation is desired at a fast rate, e.g., in an automobile radiator. As the second application problem, this will be solved using the **fully implicit** method for the partial differential equation in cylindrical geometry.

Many iterative-convergence methods have been so far developed, e.g., in finding real roots of an equation, in solving boundary value problems and psychrometric properties etc. It is interesting to couple this idea with any numerical method to solve an apparently unsolvable problem. In puffing pretreated rough rice by heated sand, the value of the heat transfer coefficient at the sand–grain surface is very important to know. If this value is known *a priori*, optimum puffing temperature and/or puffing time can be estimated. On the other hand, if the thermophysical properties of sand, product, and air and shape of the product, puffing temperature, puffing time, etc., are known,

the value of the heat transfer coefficient can be estimated by the iterative-convergence procedure. The same value can be applied to other products undergoing the same process and having similar configuration. This is the third application problem that will be solved using the **fully implicit** method for partial differential equation in slab, cylindrical, or spherical geometry.

The fourth application problem is about the universally used mechanical separator without any moving part, that is, cyclone separator. A nonlinear second-order differential equation with appropriate initial and boundary conditions, which traces the expanding helical path of a particle inside a cyclone, will be solved using any of the three fourth-order methods developed in Chapter 3.6.

The fifth and the last problem is very typical and familiar to a food or an agricultural engineer. It is about a grain drying simulation in a continuous mechanical dryer. This problem will be solved in detail using both the thin-layer and the diffusion approaches for a cylindrical or a spherical geometry when the continuous mechanical dryer is a concurrent one. The analogy between heat and mass transfer will be discussed in light of the direct use of previously developed methods for the transfer of heat to the transfer of mass.

5.1. Air Psychrometrics

All the psychrometric properties of air have been calculated considering that moist air (dry air and water vapor mixture) follows the perfect gas laws. Humidification and dehumidification involve transfer of mass between a pure liquid phase and a gas that is insoluble in the liquid. In the drying of any biological material, heat and mass transfer occur together, and concentration as well as temperature change simultaneously. The psychrometry essentially means the thermodynamic properties of the air-vapor mixture, such as, dry and wet bulb temperatures, absolute and relative humidities, vapor pressure, saturation vapor pressure, specific volume, enthalpy, latent heat of evaporation, humid heat, etc.

There are analytical as well as regression equations for the calculation of such properties. Sometimes these equations are implicit, meaning that the properties cannot be calculated directly using these equations. Numerical methods are to be employed in such cases. On the other hand, psychrometric charts are limited to a specific atmospheric pressure and to human error in reading data from them. These also cannot be used in any process simulation where a specific property is to be calculated from any other two properties given.

According to the Research Report from the Michigan State University (1974), some combinations such as wet bulb temperature and relative humidity, wet bulb temperature and specific volume, wet bulb temperature and enthalpy, specific volume and enthalpy, relative humidity and specific volume, and relative humidity and enthalpy do not have any solutions. In a

psychrometric chart, if the curves for dry bulb vs. wet bulb temperatures and dry bulb vs. relative humidity are drawn with the same scale, it is easy to find the relation between wet bulb and relative humidity without actually solving for them. The chart is useful in this respect.

In the following section, main psychrometric relations will be given in the form of equations that have been written as FORTRAN FUNCTIONS and listed in Appendix A.5.1 (**PSYCHRO.FOR**). Some of the equations are absolute with respect to units and some are in SI units but an easy method is to convert the input parameters to SI and the results to the other two units, namely, British and Metric, letting the equations remain undisturbed. The limitation cited in the literature can be overcome by employing numerical techniques of iteration–convergence. These specific cases will be discussed in detail.

All together, thirteen properties of moist air are considered, such as, dry bulb temperature (**TD**), wet bulb temperature (**TW**), absolute humidity (**HU**), relative humidity (**RH**), specific volume (**VH**), vapor pressure of water (**PV**), saturation vapor pressure (**PS**), enthalpy (**EN**), latent heat of evaporation (**LH**), density (**RO**), dew point temperature (**TP**), humid heat [specific heat of air–vapor mixture (**QH**)], and absolute viscosity (**MU**).

Thirteen variables, taken in two's, have 78 different combinations, not considering the reverse orders, meaning that the combination of dry and wet bulb is essentially the same as wet and dry bulb temperature combination. Among these 78 combinations, 12 do not have any solutions owing to the fact that each of these combinations is a function of only one independent variable. For example, the saturation vapor pressure is a function of only temperature, thus the combination of temperature and saturation vapor pressure does not have any solution because the rest of the properties cannot be calculated from this combination.

There are some psychrometric packages that identify each combination by a unique number, and the user is asked to choose a specific combination by this number. It is much more convenient to use two-letter abbreviations of the properties instead of numbers. The program developed here shows this technique. It also allows its users to give inputs in any order they want. In the same program, provision has been made to include other air–vapor mixtures used in the chemical and food industries, such as air–alcohol, air–benzene, air–toluene, etc. The complete program is listed in Appendix A.5.2 as **PSY2000.FOR**.

As has been already mentioned, the program uses all the equations in their absolute forms or in SI units, but it can take input in any of the three unit systems, namely, SI, Metric (MKS) or British (FPS). Instead of modifying the equations, it is convenient to convert the input data to SI and finally convert the results to the unit system initially chosen.

Psychrometric relations

1. Saturation vapor pressure, P_s , as a function of temperature
(FUNCTION PST in Appendix A.5.1)

$$P_s = 22087836.75 e^{\frac{0.01}{T+273.15} (374.136-T) \sum_{i=1}^8 F_i (0.65-0.01.T)^{i-1}} \quad 5.1.1$$

where

P_s : saturation vapor pressure, Pa

T : temperature, °C

$F_1 = -741.9242$, $F_2 = -29.721$, $F_3 = -11.55286$,

$F_4 = -0.8685635$, $F_5 = 0.1094098$, $F_6 = 0.439923$,

$F_7 = 0.2520658$, $F_8 = 0.05218684$

This expression is given by Keenan et al. (1969) for a temperature range of 0 to 374°C.

2. Absolute humidity, H_u , as a function of dry and wet bulb temperatures
(FUNCTION HDBWB in Appendix A.5.1)

$$H_u = \frac{(h_c - (C_{pl} - C_{pv})T_w)H_u^* - C_{pa}(T_d - T_w)}{h_c + C_{pv}T_d - C_{pl}T_w} \quad 5.1.2$$

where

H_u : absolute humidity, kg moisture/kg dry air

T_d and T_w : dry and wet bulb temperatures, °C

C_{pl} : specific heat of liquid water = 4186.8 J/kg.°C

C_{pa} : specific heat of dry air = 1004.832 J/kg.°C

C_{pv} : specific heat of water vapor, 1858.9392 J/kg.°C

h_c : heat capacity of moist air = 2500915.2 J/kg

H_u^* : absolute humidity at wet bulb temperature

3. Relative humidity, R_h , as a function of vapor pressures
(FUNCTION RHHT in Appendix A.5.1)

$$R_h = \frac{P_v}{P_s} \quad 5.1.3$$

where

R_h : relative humidity, fraction

P_v : partial vapor pressure at dry bulb temperature, Pa

P_s : saturation vapor pressure at dry bulb temperature, Pa

4. Vapor pressure, P_v , as a function of absolute humidity
(FUNCTION PVH in Appendix A.5.1)

$$P_v = \frac{P \cdot H_u}{H_u + \frac{W_{mv}}{W_{ma}}} \quad 5.1.4$$

where

P_v : partial vapor pressure, Pa

P : atmospheric pressure, Pa

W_{mv} : molecular weight of water vapor = 18.051534

W_{ma} : molecular weight of air = 28.9645

5. Humid volume, V_h , as a function of temperature and absolute humidity
(FUNCTION VTH in Appendix A.5.1)

$$V_h = \frac{R(T + 273.15)(1 + \frac{W_{ma}}{W_{mv}} \cdot H_u)}{P} \quad 5.1.5$$

where

V_h : humid volume, m³ moist air/kg dry air

R : gas constant for dry air = 287.055724249 J/kg.K

6. Density, R_o , as a function of humid volume and absolute humidity
(FUNCTION ROTH in Appendix A.5.1)

$$R_o = \frac{1 + H_u}{V_h} \quad 5.1.6$$

where

R_o : density, kg moist air/m³ moist air.

7. Enthalpy, E_n , as a function of temperature and absolute humidity
(FUNCTION ENTH in Appendix A.5.1)

$$E_n = C_{pa} \cdot T + H_u(h_c + C_{pv} \cdot T) \quad 5.1.7$$

where

E_n : enthalpy, J/kg.

8. Latent heat of evaporation, L_h , as a function of temperature
(FUNCTION HLT in Appendix A.5.1)

- a. For a temperature range of 459.69°R to 491.69°R

$$L_h = 1220.884 - 0.05077(T_R - 459.69) \quad 5.1.8.a$$

- b. For a temperature range of 491.69°R to 609.69°R,

$$L_h = 1075.8965 - 0.56983(T_R - 459.69) \quad 5.1.8.b$$

c. For a temperature range higher than 609.69°R,

$$L_h = \sqrt{1354673.214 - 0.9125275587.T_R^2} \quad 5.1.8.c$$

where

L_h : latent heat of vaporization, Btu/lb

T_R : temperature, °R.

To convert L_h to SI unit, that is, J/kg, it is multiplied by the conversion factor 2326.0 [1 Btu/lb = 2326.0 J/kg].

9. Humid heat, Q_h , as a function of absolute humidity
(FUNCTION HQH in Appendix A.5.1)

$$Q_h = C_{pa} + H_u.C_{pv} \quad 5.1.9$$

where

Q_h : humid heat, J/kg.°C.

10. Absolute viscosity M_u , as a function of temperature
(FUNCTION VST in Appendix A.5.1)

$$M_u = (1.7471 + 4.0181 \times 10^{-3}.T)10^{-5} \quad 5.1.10$$

where

M_u : absolute viscosity, Pa.s

All the rest of the functions in Appendix A.5.1 are derived from the relationships given by Eqs. (5.1.1) through (5.1.10).

Calculation of the psychrometric properties

Each of the 66 combinations (78 - 12) is ultimately reduced to solving either the combination of T_d and T_w or T_d and H_u . The first combination is explicit where no iteration-convergence method is required, whereas the second is implicit. Let us consider them one by one:

1. T_d and T_w are given, and P is known.

Saturation vapor pressure is calculated by Eq. (5.1.1),

Absolute humidity is calculated by Eq. (5.1.2),

Relative humidity is calculated by Eq. (5.1.3),

Vapor pressure is calculated by Eq. (5.1.4),
 Specific volume is calculated by Eq. (5.1.5),
 Density is calculated by Eq. (5.1.6),
 Enthalpy is calculated by Eq. (5.1.7),
 Latent heat of evaporation is calculated by Eq. (5.1.8),
 Humid heat is calculated by Eq. (5.1.9), and
 Viscosity is calculated by Eq. (5.1.10).

Dew point temperature is defined as the temperature at which the air-vapor mixture must be cooled to become saturated without altering the absolute humidity of it. Thus, T_p is first assumed as $(T_d - 5)$ and saturation vapor pressure is calculated by Eq. (5.1.1) as P_{v1} using this assumed value of T_p . This calculated value is then compared with the actual value of P_v . If the absolute difference between these two values does not fall below a small error limit of 0.05, the process is repeated with the corrected value of T_p as $[T_p + (P_v - P_{v1})/P_v]$.

2. T_d and H_u are given, and P is known.

The wet bulb temperature is first assumed as $(T_d - 5)$, and, with this assumed value, absolute humidity is calculated by Eq. (5.1.2) as H_{u1} . The absolute difference between H_u and H_{u1} is then compared with a small error limit of 10^{-6} . This process is repeated with the corrected value of T_w as $[T_w + (H_u - H_{u1})/H_u]$ until the convergence criterion is met. The problem now becomes equivalent to the previous one just solved.

Let us now discuss the other cases that do not fall into this format. In the event of any clarification about any of the combinations, subroutines **VALUES**, **SELECT** and **PROP** should be consulted.

3. P_s and any one parameter other than T_d , L_h , Q_h , and M_u are given.

First, a dry bulb temperature range is selected between T_1 and T_2 . Using Eq. (5.1.1), P_{s1} and P_{s2} are calculated at these two temperatures, respectively. In the program, these two temperatures have been initially assumed as 20 and 50°C, respectively. With these values, if the convergence does not occur, the user is asked to give any other values.

In Figure 5.1.1, it is shown that the value of the actual P_s given, can be on either side of the saturation vapor pressure calculated by the assumed temperatures T_1 and T_2 . From similar triangles, the extrapolated distance **a** from the origin can be obtained as

$$a = \frac{P_{s2} \cdot T_1 - P_{s1} \cdot T_2}{T_2 - T_1}$$

Now, once taking a and P_{s1} , from similar triangles, the unknown, T , is calculated as

$$T = \frac{(a + P_s)T_1}{a + P_{s1}} \quad 5.1.11$$

and then considering a and P_{s2} , from the relevant similar triangles, T is calculated as

$$T = \frac{(a + P_s)T_2}{a + P_{s2}} \quad 5.1.12$$

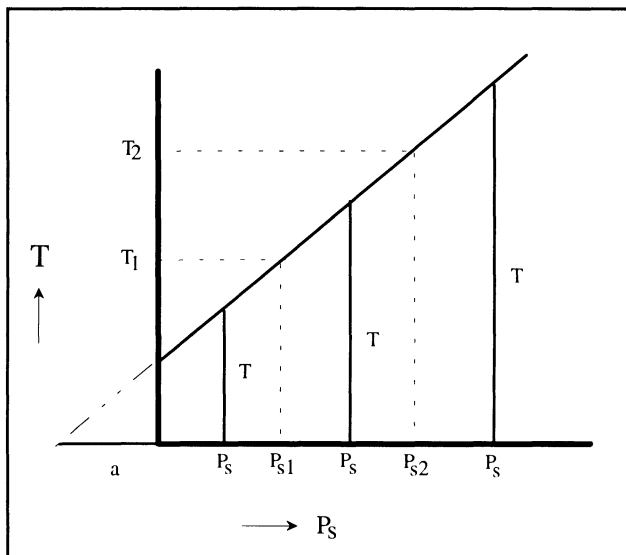


Figure 5.1.1. Calculation of temperature from saturation vapor pressure.

Finally an arithmetic average is made of these temperatures obtained from Eqs. (5.1.11) and (5.1.12). With this average temperature, saturation vapor pressure P_{s1} is calculated by Eq. (5.1.1) and temperature T_d is corrected by $[T_d + (P_s - P_{s1})/P_s]$. This process is continued until the absolute difference between P_s and P_{s1} falls below an error limit of 0.01. Subroutine **TDS** is called when this procedure is required.

4. L_h and one parameter other than T_d , P_s , Q_h , and M_u are given.

Here the procedure is very similar to the one just given in combination 3. Instead of saturation vapor pressure on the x-axis, latent heat of evaporation is used. It is calculated by Eq. (5.1.8).

The six combinations of two properties mentioned earlier that were considered unsolvable for the psychrometric properties have been solved and the solution of one such combination, e.g., relative humidity and enthalpy is demonstrated here. Other combinations are solved in similar manners and can be clarified by studying the subroutine **SELECT**.

5. R_h and E_n are given, and P is known.

First an arbitrary value of T_d is chosen, and absolute humidity is calculated by the following equation obtained from Eq. (5.1.7):

$$H_u = \frac{E_n - C_{pa} \cdot T_d}{h_c + C_{pv} \cdot T_d} \quad 5.1.13$$

Then an estimation of relative humidity R_{hl} is made by the following equation obtained from Eqs. (5.1.3) and (5.1.4):

$$R_{hl} = \frac{P \cdot H_u}{P_s(H_u + \frac{W_{mv}}{W_{ma}})} \quad 5.1.14$$

The absolute difference between the given and the estimated relative humidity values is checked against a small error limit of 10^{-6} . Until this convergence criterion is met, at each iteration the value of T_d is corrected by $[T_d + (R_{hl} - R_h)/R_h]$. Once the convergence criterion is satisfied, it becomes a problem of solving with the combination of T_d and H_u . The flow diagram shown in Figure 5.1.2 will make this scheme clear.

In the program **PSY2000.FOR**, there are 5 subroutines, namely, **TITLE**, **VALUES**, **SELECT**, **PROP**, and **TDS**, along with 18 multiline functions. The main program reads the value of the atmospheric pressure and the two-letter abbreviations of any two properties. After it writes those abbreviations on the screen and asks the user to give corresponding values in any one of the three unit systems chosen, it calls the subroutine **VALUES**, which initializes all the properties to -1.0 except the ones chosen and converts the properties given to SI units. Then the subroutine **SELECT** is called inside a loop. It picks up the nonnegative properties and calculates some other properties with the help of appropriate equations, and in some cases, by iterative methods to streamline the problem to a simple format. It then calls the subroutine **PROP** to solve for the rest of the properties. In this **PROP** that the final results are converted to appropriate units asked by the user. Table 5.1.1 shows the units of each variable in three unit systems used in the program PSY2000 and a typical computer output by the same in SI units is presented in Table 5.1.2.

The program **PSY2000.FOR** has been developed in such a way that it allows its user to write the results in a defined file.

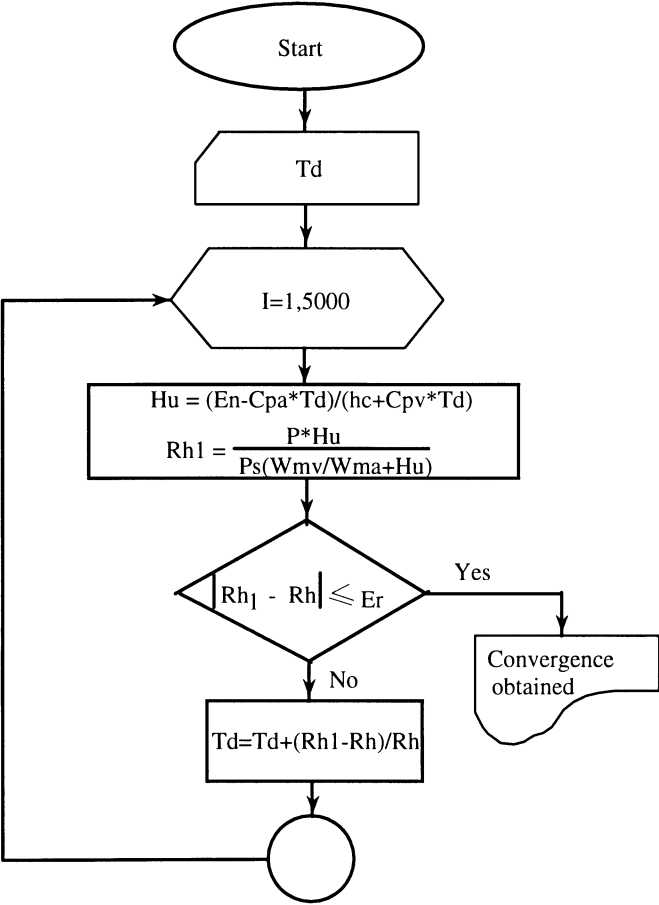


Figure 5.1.2. Flow diagram of convergence scheme in combination 5.

Table 5.1.1. Units in different systems.

	SI	Metric	British
Temperature	°C	°C	°F
Pressure	Pa	mm mercury	psi
Enthalpy/L. heat	J/kg	kcal/kg	Btu/lb
Humid heat	J/kg.°C	kcal/kg.°C	Btu/lb.°F
Density	kg/m ³	kg/m ³	lb/ft ³
Sp. volume	m ³ /kg	m ³ /kg	ft ³ /lb
Viscosity	Pa.s	cP	lb/ft.s

Table 5.1.2. Calculated properties of air–water vapor system.
Atmospheric pressure = 101325.0 Pa

Td	Tw	Tp	Hu	Pv	Ps	Rh	Vh
30.00	20.00	14.82	0.010519	1685.19	4245.80	39.6907	0.873355
En		Lh	Qh		Mu	Ro	
57039.78		2388548.71	1024.3869		0.0000186764	1.157054	
Properties given : TD = 3.000000D+01, TW = 2.000000D+01.							

5.2. Tubular Heat Exchanger with Annular Fin of Uniform Thickness.

Heat transfer in cylindrical geometry has been covered in detail in Chapter 3.7. The principal equation of heat transfer considered so far did not contain any heat source/sink term. Fins on the outer surface of a heat exchanger tube act as a sink or negative heat source because they help in dissipating heat to the atmosphere. Figure 5.2.1 shows the section of an annular fin.

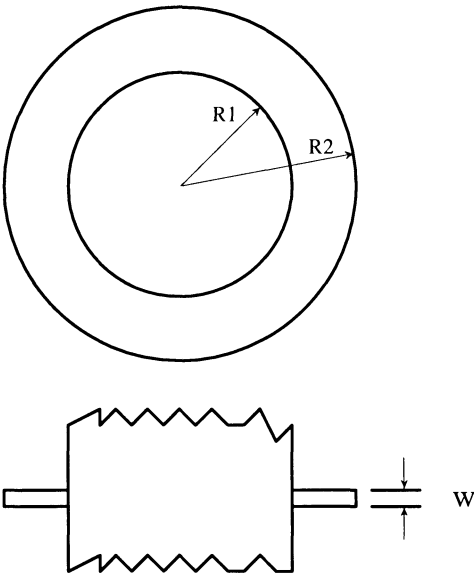


Figure 5.2.1. Cross sectional view of an annular fin.

The governing heat transfer equation with a heat source term looks typically as given below:

$$\rho \cdot C_p \cdot A \cdot \frac{\partial T}{\partial t} = k \cdot \frac{\partial}{\partial r} \left(A \cdot \frac{\partial T}{\partial r} \right) + h \cdot P (T_\infty - T) \quad 5.2.1$$

where

P : perimeter of fin cross section, m

T_∞ : surrounding temperature, °C

h : convective heat transfer coefficient at the fin surface, $W/m^2 \cdot ^\circ C$.

The extra term on the right hand side of Eq. (5.2.1) is the heat source term. When T_∞ is greater than the temperature of the fin, the fin is gaining heat from the surroundings. This term is negative ($T > T_\infty$) when the fin is dissipating heat to its surroundings as a heat source.

Now, Eq. (5.2.1) can be written using Eq. (3.7.41) with Eqs. (3.7.42) and (3.7.43) in Chapter 3.7 as the following,

$$\rho \cdot C_p \cdot A_j \cdot \frac{T_{i+1,j} - T_{i,j}}{\Delta t} = \left[k \cdot \frac{T_{i+1,j+1} - T_{i+1,j}}{\Delta r^2} \cdot A_{j+1/2} - k \cdot \frac{T_{i+1,j} - T_{i+1,j-1}}{\Delta r^2} \cdot A_{j-1/2} \right] + h \cdot P (T_\infty - T_{i+1,j}) \quad 5.2.2$$

The perimeter, P , should take care of both the sides of the fin exposed. Thus

$$P = 2(2\pi \cdot r_j) = 4\pi \cdot r_j$$

The fin should be thin enough to allow its maximum number thereby offering a maximum surface area for heat dissipation. In Figure 5.2.1., it has been shown as w . Thus segment areas and volumes in cylindrical geometry are expressed as

$$A_j = 2\pi[r_2 - (j-1)\Delta r]w$$

$$V_j = \pi \cdot \Delta r[2r_2 - (2j-1)\Delta r]w$$

Rearranging Eq. (5.2.2),

$$C_1 \cdot T_{i+1,j-1} + C_2 \cdot T_{i+1,j} + C_3 \cdot T_{i+1,j+1} = DD \cdot T_{i,j} + S \cdot T_\infty \quad 5.2.3$$

where

$$DD = -\frac{\Delta r^2}{\alpha \cdot \Delta t}$$

$$C_1 = \frac{A_j + A_{j-1}}{2 \cdot A_j}$$

$$C_2 = DD - (C_1 + C_3) + S$$

$$C_3 = \frac{A_{j+1} + A_j}{2 \cdot A_j}$$

$$S = \frac{h \cdot P \cdot \Delta t \cdot DD}{\rho \cdot C_p \cdot A_j}$$

Now, the nondimensional term, S , can be incorporated in the heat transfer equations developed so far to make them more general, and common heat transfer without any heat source can be had by letting $S = 0$.

Formulation of the problem

The tube having the annular fin around it carries a hot fluid, and the base temperature of the fin is maintained at T_b , whereas the outside fluid temperature around the fin is at T_∞ . In the program **FIN.FOR** listed in Appendix A.5.4, it has been denoted by TC and the base temperature as TB. This problem is identical to one with side 1 convective and side 2 at constant temperature. In this case, side 1 represents the outside of the fin and side 2 its base. Thus the following matrix, similar to the one given by Eq. (3.7.62) in Chapter 3.7, describes the equations to solve for the unknown temperatures at one advanced step in time:

$$\begin{bmatrix} (C_2 + CC_0) & C_3 & & & \\ C_1 & C_2 & C_3 & & \\ & C_1 & C_2 & C_3 & \\ \dots & & & & \\ & & C_1 & C_2 & C_3 \\ & & & C_1 & C_2 \end{bmatrix} \begin{bmatrix} T_{i+1,2} \\ T_{i+1,3} \\ T_{i+1,4} \\ \dots \\ T_{i+1,n-1} \\ T_{i+1,n} \end{bmatrix} = \begin{bmatrix} DD \cdot T_{i,2} + S \cdot T_\infty - CC \\ DD \cdot T_{i,3} + S \cdot T_\infty \\ DD \cdot T_{i,4} + S \cdot T_\infty \\ \dots \\ DD \cdot T_{i,n-1} + S \cdot T_\infty \\ DD \cdot T_{i,n} + S \cdot T_\infty - C_3 \cdot T_b \end{bmatrix} \quad 5.2.4$$

where $T_{i+1,1}$ is given by Eq. (3.7.60) as

$$T_{i+1,1} = \frac{T_\infty + DD1 \cdot T_{i+1,2}}{1 + DD1}$$

and DD1 is given by Eq. (3.7.33). Expressions for CC and CC_0 (see Chapter 3.7) are given by

$$CC = \frac{T_\infty \cdot C_1}{1 + DD1} ; \text{ and } CC_0 = \frac{DD1 \cdot C_1}{1 + DD1}$$

The fin problem is solved using matrix (5.2.4), which is very similar to matrix (3.7.62). Subroutine **TRIDGL.FOR** listed in Appendix A.3.7.3 is used to solve the tridiagonal matrix of the implicit scheme. The implicit scheme is preferred due to its unconditional stability, which was discussed with an example in Chapter 3.7.

The following input data were used in the program **FIN.FOR**. Its output is shown in Table 5.2.1.

Input data

Initial temperature,	$T_0 = 25\text{ }^{\circ}\text{C}$
Base temperature,	$T_b = 200\text{ }^{\circ}\text{C}$
Surroundings temperature,	$T_{\infty} = 25\text{ }^{\circ}\text{C}$
Base radius,	$r_1 = 7.5\text{ cm}$
Tip radius,	$r_2 = 12.5\text{ cm}$
Width,	$w = 0.5\text{ cm}$
Heat transfer coefficient,	$h = 60\text{ W/m}^2\cdot^{\circ}\text{C}$
Specific heat,	$C_p = 400\text{ J/kg}\cdot^{\circ}\text{C}$
Thermal conductivity,	$k = 40\text{ W/m}\cdot^{\circ}\text{C}$
Density,	$\rho = 8825\text{ kg/m}^3$
Time step,	$\Delta t = 0.5\text{ s}$
Total time exposure,	$t = 5\text{ min}$

The density, the specific heat, and the thermal conductivity are supplied by the user through a program segment **THERMO.FOR**, which is included in the main program. It is listed in Appendix A.5.3. The program stops when the steady-state condition is reached or total time of exposure has elapsed without reaching the steady state. In the latter case, it gives a message that the steady state has not reached and that the user should increase the limit on the total time of exposure. Once the steady state is reached, it calculates the rate of heat dissipation by the fin and prints it along with the time to reach this and the mean temperature of the fin.

The steady state is decided to have been reached, when the absolute difference between the temperatures at the same nodal point but at two consecutive time steps falls below a predetermined small value. In the program, this small value has been kept as 0.001 and the n -th nodal point has been selected. It could have been anywhere in the fin except at its base where the temperature does not change with time. This condition is checked starting the third step in time when a noticeable change in temperature starts showing up.

The temperature profile at steady-state condition is shown in Table 5.2.1. A total of 20 nodal points was selected along the direction of heat transfer.

The program is dimensioned for a maximum number of 100 such nodal points.

Table 5.2.1. Computer output for the fin problem (at steady state).

Mean Temperature = 132.78 C at Time = 264.50 s					
108.10	108.42	109.07	110.06	111.41	113.12
115.22	117.72	120.64	124.00	127.83	132.16
137.01	142.43	148.45	155.11	162.47	170.57
179.48	189.27	200.00			
[Rate of HEAT dissipated = 411.32078 W]					

A finned tube heat exchanger can be designed on the analysis of one fin shown here. If the total heat dissipation rate to be achieved by the heat exchanger is known, the number of fins can be calculated from the total heat dissipated from one such fin, which is 411.3 W in this case. For example, to dissipate 5 kW, 13 such fins will be required, which is obtained by dividing 5000 W by 411.3 W and adding 1 for the positive fraction on the division. In this particular case, it took 264.5 s to reach the steady-state condition.

5.3. Heat Transfer Coefficient in Puffing of Rice by Heated Sand

Heated sand is used as an efficient heat transfer medium in many physical processes, and it is important to have a precise knowledge of its heat transfer characteristics. When pretreated rough rice is roasted in heated sand, the vapor pressure inside the grain rapidly builds up and finally attains a limit where the grain cannot hold it any more, thus letting the pressure come out of it with an explosion, and the rough rice gets puffed. The pretreatments on rough rice are meant to harden its outer surface, thereby making it capable of holding vapor pressure higher than atmosphere and causing explosion during the heat treatment by heated sand.

The critical vapor pressure for the explosion is related directly to the average temperature of the grain during puffing, and, in turn, the average temperature and the time to attain it depend on the heat transfer coefficient at the interface of the grain and the sand. The heat transfer coefficient mainly is the property of the fluid and the configuration (shape and surface properties) of the solid. Thus once estimated, its value can be used for puffing of other materials that have a similar configuration. A precise estimation of the heat transfer coefficient makes it possible to optimize the puffing process of other products as well.

Research conducted by Das and Srivastav (1989) is limited to the use of a one-term analytical approximation of the average temperature for a cylindrical geometry. It suffers the inevitable inaccuracy in estimating the heat transfer coefficient from the slope of a regression line drawn through some points obtained experimentally. The method developed here uses a numerical approach and eliminates the limitations of the former approach.

Formulation of the problem

Rough rice of a long grain variety was considered as an infinite cylinder. The puffing time observed was short, and the following assumptions were made:

1. the moisture content of rice remains constant until puffing,
2. the density [ρ], thermal conductivity [k], thermal diffusivity [α], and specific heat [C_p] of rice do not change during puffing. Their values should be calculated at the arithmetic mean of the initial and the average temperature of the grain,
3. the rice puffs when its average temperature reaches a critical value,
4. the temperature of the sand during puffing remains constant, and
5. the heat transfer coefficient [h] remains constant.

The outside of the rice grain is considered to have the convective boundary condition (Robbins condition) and its center to have the derivative condition due to symmetry (Neumann condition). Thus for the implicit method, the matrix represented by Eq. (3.7.64) in Chapter 3.7 is solved. The complete program **PUFF.FOR** is listed in Appendix A.5.7.

Here, in the method developed, rough rice or any other product the shape of which can be defined by an infinite slab, an infinite cylinder, or a sphere, may be considered. The implicit method developed in the earlier chapter, which is unconditionally stable irrespective of the size of the time step, is used to estimate the average temperature of the rice grain. A guess for the heat transfer coefficient, h_0 , is first supplied. The average temperature is calculated by solving Eq. (3.7.64). This temperature is checked with the measured one. If the absolute difference between the estimated and the measured average temperatures, T_{mean} and T_{av} , respectively, does not fall below a predetermined small value of 0.01°C , the heat transfer coefficient is corrected by the following expression and the loop continues :

$$h = h_0 + T_{\text{av}} - T_{\text{mean}} \quad 5.3.1$$

This scheme, given by Eq. (5.3.1), is found to have rapid convergence. If the initial guess is not too far off, it converges within 10 iterations. The

maximum number of iterations permitted has been given as 100 in the program. To calculate the area and volume segments of the configuration considered, the program calls the subroutine **GEOMTY** listed in Appendix A.3.7.2. At each iteration, it calls the subroutines **COEFF** (listed in Appendix A.5.6) and **TRIDGL** (listed in Appendix A.3.7.3) and calculates the average temperature.

The program when compiled and run, gives an option of defining the product configuration, such as slab, cylindrical, or spherical. Then it asks for the inputs, such as, initial, outside, and average product temperature; product moisture content in percent wet basis; and the first guess of the heat transfer coefficient. Then, it asks for the principal dimension of the product (thickness for the slab and radius for the other two geometries) in cm, total processing time in s, and number of divisions/nodes. The program is dimensioned for a maximum of 100 such divisions along the principal dimension of the product. Finally, the program asks for the time step showing its upper limit for the explicit method. For the implicit scheme used, this step can be up to 10 times as large without adversely affecting the accuracy. If the scheme does not converge within 100 iterations, the control goes back to the main menu, letting the user give a different guess on the heat transfer coefficient, and the process starts all over again.

The results show the mean temperature of the product calculated within the error limit of 0.01, the value of the calculated heat transfer coefficient at the convergence criterion, and the number of iterations for convergence. The thermophysical properties of the product are supplied by the program segment **THERMO1.FOR**, which should be provided by the user and is included at the end of the program **FIN.FOR**. It is listed in Appendix A.5.5. The properties include (i) density of pure water, ρ_w , as a function of temperature, (ii) specific heat of pure water, C_{pw} , as a function of temperature, (iii) thermal conductivity of water, k_w , as a function of temperature, (iv) density of the product, ρ_p , as a function of water fraction and its property, (v) specific heat of the product, C_{pp} , as a function of water fraction and its property, and (vi) thermal conductivity of the product, k_p , as a function of water fraction and its property. The thermal diffusivity of the product, α_p , is calculated as

$$\alpha_p = \frac{k_p}{\rho_p \cdot C_{pp}}$$

For simplicity, the properties, such as density and specific heat, are given as constant. The user should express these properties as functions of temperature and moisture content. An example of this has been shown by the functions $KW(T)$ and $KP(T, XM)$ for thermal conductivity of the pure water and the product, respectively, in **THERMO1.FOR**.

Input data

Initial product temperature,	$T_0 = 30\text{ }^{\circ}\text{C}$
Sand temperature,	$T_c = 250\text{ }^{\circ}\text{C}$
Average product temperature during puffing (experimental),	$T_{av} = 155\text{ }^{\circ}\text{C}$
Product moisture,	$X_m = 10\text{ \% wet basis}$
Product radius,	$R = 1.135\text{ mm}$
Processing/puffing time (experimental),	$t = 15\text{ s}$
Initial guess on htc,	$h_0 = 70\text{ W/m}^2\text{.}^{\circ}\text{C}$
Number of nodes,	$n = 50$
Time step,	$\Delta t = 0.01\text{ s}$

With these input data, at the calculated average product temperature of $155\text{ }^{\circ}\text{C}$, the heat transfer coefficient is estimated as $88.7\text{ W/m}^2\text{.}^{\circ}\text{C}$. It needed five iterations to converge.

5.4. Residence Time of a Particle Inside a Cyclone Separator

Nonfiltration air/gas cleaning techniques using cyclones are common in the industries. The simplest type of dust trapping device is a settling chamber, however, the efficiency of separation of such units is very low. Cyclones, on the other hand, are the most widely used type of separating equipment and are simple in construction, have no moving parts and thus require little or no maintenance.

The centrifugal force on particles in a swirling air/gas stream is much greater than gravity and thus cyclones are effective in the removal of particles up to $10\text{ }\mu\text{m}$ diameter when their specific gravity is unity (Strauss, 1974). Solid particles may vary in diameter and specific gravity and are not necessarily spherical. For the mathematical modeling of the process, dimensions of the particles are converted to equivalent spheres to reduce the complexity of the analysis. In Figure 5.4.1 a schematic of particle separation inside a cyclone is shown.

Formulation of the problem

Determining the trace of the path of an expanding helix inside a cyclone requires Newton's second law of motion in its differential form and Stoke's law to yield a nonlinear differential equation, which can only be solved by employing numerical methods.

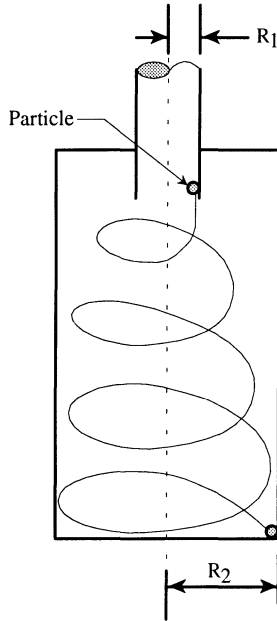


Figure 5.4.1. Trace of a particle path inside the cyclone.

Strauss (1974) developed a nonlinear second-order differential equation in its dimensionless form. Later Chandra and Payne (1986) expressed it in its original form as

$$\frac{d^2 r}{dt^2} + C_1 \cdot \frac{dr}{dt} - \frac{C_2^2}{r^3} = 0 \quad 5.4.1$$

where

$$C_1 = \frac{9\mu}{2r_p^2(\rho_p - \rho)} \quad 5.4.2$$

and

$$C_2 = \omega \cdot R_1^2 \quad 5.4.3$$

r : variable radius of the particle trajectory, m

ω : angular velocity at $r = R_1$, rad/s

μ : absolute viscosity of air, Pa.s

ρ : density of air, kg/m³

ρ_p : density of particle, kg/m³

r_p : radius of particle, m

In order to solve Eq. (5.4.1), we need to split it into two first-order ordinary differential equations as follows:

$$\frac{dr}{dt} = z \quad 5.4.4$$

$$\frac{dz}{dt} = -C_1 \cdot z + \frac{C_2^2}{r^3} \quad 5.4.5$$

The initial conditions for Eqs. (5.4.4) and (5.4.5) are

$$r(t = 0) = R_1 \text{ and } z(t = 0) = 0, \text{ respectively.}$$

The subroutine **RKPC** developed in Chapter 3.6 and listed in Appendix A.3.6.5 as **RKPC.FOR** for solving ordinary differential equations is employed to solve Eqs. (5.4.4) and (5.4.5). The main program, **CYCLONE.FOR**, which calls this subroutine, is listed in Appendix A.5.8. The input data to this program is shown below:

Input data

Throat diameter of cyclone,	$D_1 = 7 \text{ cm}$
Inside diameter of cyclone,	$D_2 = 15 \text{ cm}$
Inlet fan width,	$F_w = 7 \text{ cm}$
Inlet fan height,	$F_h = 7 \text{ cm}$
Particle diameter,	$d_p = 20 \text{ }\mu\text{m}$
Particle density,	$\rho_p = 1000 \text{ kg/m}^3$
Atmospheric pressure,	$P = 101325 \text{ Pa}$
Dry bulb temperature of air,	$T_0 = 25 \text{ }^\circ\text{C}$
Wet bulb temperature of air,	$T_w = 18 \text{ }^\circ\text{C}$
Air flow at fan inlet,	$A_f = 0.05 \text{ m}^3/\text{s}$
Time step,	$\Delta t = 10^{-4} \text{ s}$
Time of residence (stopping criterion),	$t_r = 0.5 \text{ s}$

With the above input data, the program calculates absolute humidity, density, and absolute viscosity of air using the psychrometric functions listed in Appendix A.5.1. Then it computes the number of time steps, air inlet velocity, and angular velocity at the inlet and C_1 and C_2 given by Eqs. (5.4.2) and (5.4.3), respectively. The subroutine **START** is then called and subsequently subroutine **RKPC**.

At each time step, the variable radius of the particle trajectory calculated is compared against R_2 , that is, the destination radius. The program stops when the computed radius of the particle trajectory is greater than or equal to

R_2 . When this occurs, the residence time, t , is corrected by the following expression:

$$t = \frac{t_c \cdot R_2}{r_c}$$

where

t_c : estimated residence time, s

r_c : estimated radius of particle trajectory after time t_c , m.

If the estimated residence time is more than the stopping criterion given as input, the program stops giving a message that it is not sufficient for the particle to settle inside the cyclone, and the control goes back to the main menu.

A rough estimation of the length of the trajectory, or, in other words, the settling length for the particle, is made by assuming the average radius of the trajectory as the arithmetic mean of R_1 and R_2 and the thickness and pitch of the air spiral inside equal to the fan height, F_h . Thus, the number of revolutions of the spiral,

$$n = \frac{\omega}{2 \cdot \pi}$$

where

$$\omega = \frac{V_{air}}{\frac{R_1 + R_2}{2}}$$

V_{air} : velocity of air at the cyclone inlet, m/s.

Now, if the pitch of the spiral is taken equal to the fan inlet height, the distance that the particle will move before touching the cyclone surface at its outer radius can be calculated by $(n \cdot t \cdot F_h)$.

With the above input data, the residence time of the particle inside the cyclone is estimated as 0.12 s and the settling length as 24.34 cm.

5.5. Drying Simulation using Diffusion and Thin-Layer Concepts

In Chapter 3.7, all the equations are derived for heat transfer with a note that these are strikingly similar to mass transfer and are equally applicable to it. Before solving the present problem, these transformations and similarities will be discussed in detail.

In the continuous drying operation, heated air is used as the drying medium. When heated air enters the first layer of the grain, its temperature and relative humidity are known. However, as it passes through the grain layers, it picks up moisture and loses heat on the way, thus decreasing its temperature and increasing its relative humidity. If the entire grain bed is

divided into many thin imaginary layers, the condition of the air entering the second layer will be very different from that entering the first. The mechanical grain drying simulation should keep track of these changes in the air and in the grain properties at each layer so that it depicts the real situation closely.

Formulation of the problem

Overall mass balance on the water parts of the air and of the grain results in the following differential equation:

$$\frac{dH}{dx} = -\frac{G_p}{G_a} \cdot \frac{dM}{dx} \quad 5.5.1$$

Overall energy balance on the fluid phase (air) yields the following:

$$\frac{dT_a}{dx} = \frac{-h \cdot a_s (T_a - T_g)}{G_a (C_{pa} + C_{pv} \cdot H)} \quad 5.5.2$$

Overall energy balance on the solid phase (grain) gives:

$$\frac{dT_g}{dx} = \frac{h \cdot a_s (T_a - T_g)}{G_p (C_{pp} + C_{pw} \cdot M)} - \frac{h_{fg} + C_{pv} (T_a - T_g)}{G_p (C_{pp} + C_{pw} \cdot M)} \cdot G_a \cdot \frac{dH}{dx} \quad 5.5.3$$

where

- H : variable absolute humidity of air, kg moisture/kg dry air
- M : variable product moisture, fraction dry basis
- T_a : variable air temperature, °C
- T_g : variable grain temperature, °C
- h : heat transfer coefficient at the grain–air interface, W/m²·°C
- a_s : specific product surface area, m²/m³
- h_{fg} : latent heat of evaporation from grain surface, J/kg
- C_{pa} : specific heat of dry air, J/kg·°C
- C_{pv} : specific heat of water vapor, J/kg·°C
- C_{pw} : specific heat of liquid water, J/kg·°C
- C_{pp} : specific heat of dry product, J/kg·°C
- G_a : mass flow rate of dry air, kg/s
- G_p : mass flow rate of dry product, kg/s
- x : variable product depth, m.

The above three equations [(5.5.1) through (5.5.3)] are of first-order for concurrent flow continuous drying, and their derivation can be found in Chandra (1983). In order to simulate the drying process, there should be

another equation to describe the change of product moisture with respect to the time or space variable. There are two ways by which such an equation can be derived — one is to develop a regression equation by the drying time and corresponding moisture content data. The regression equation should be such that it can be analytically differentiated with respect to time. The second way is to use the parabolic partial differential equation in cylindrical or in spherical geometry to express the moisture gradient of the product. The first one is known as the **thin-layer equation** and the second one as the **diffusion equation**.

Thin-Layer Equation

A special exponential equation given by Page (1949) is found to fit the drying data adequately, which is given by:

$$\frac{M - M_e}{M_0 - M_e} = e^{-P \cdot t^Q} \quad 5.5.4$$

where

M_e : equilibrium moisture content, fraction db

M_0 : initial product moisture, fraction db

t : drying time, s

P and Q : regression coefficients

Page's equation (5.5.4) can be differentiated with respect to t to obtain,

$$\frac{dM}{dt} = -(M_0 - M_e)P \cdot Q \cdot t^{Q-1} \cdot e^{-P \cdot t^Q} \quad 5.5.5$$

Since the independent variable in the first three differential equations is the space variable, it will be convenient to transform Eq. (5.5.5) into the same form so that Eqs. (5.5.1), (5.5.2), (5.5.3), and the transformed (5.5.5) can be simultaneously solved using any fourth-order method of solving ordinary differential equations learned in Chapter 3.6. Thus

$$\frac{dM}{dx} = \frac{\frac{dM}{dt}}{\frac{dx}{dt}} = -\frac{(M_0 - M_e)M_r \cdot P \cdot Q \cdot t^{Q-1}}{V} \quad 5.5.6$$

where

$$M_r : \text{moisture ratio} = \frac{M - M_e}{M_0 - M_e}$$

$$V : \text{product velocity inside the dryer} = \frac{dx}{dt}, \text{ m/s.}$$

From Eq. (5.5.4), t is expressed as

$$t = \left[\frac{-\ln(M_r)}{P} \right]^{1/Q}$$

and the equivalent time used in the program is given by adding the time step, Δt with t , that is,

$$\text{Equivalent drying time} = \left[\frac{-\ln(M_r)}{P} \right]^{1/Q} + \Delta t \quad 5.5.7$$

In Eq. (5.5.6), time t should be replaced by the equivalent time expressed by Eq. (5.5.7). Now, the program **DRY2000.FOR** is developed for solving Eqs. (5.5.1), (5.5.2), (5.5.3), and (5.5.6) in conjunction with Eq. (5.5.7) by any of the three fourth-order methods, namely, Runge-Kutta, Adam's, or Hamming's Predictor-Corrector. The subroutine **RKPC** developed in Chapter 3.6 and listed in Appendix A.3.6.5 as **RKPC.FOR** is called for their solutions. The complete program listing of **DRY2000.FOR** is provided in Appendix A.5.10.

Diffusion equation

Instead of presenting the final partial differential equation for moisture diffusion, it will be derived from the first principle, and its similarity with the heat transfer equation then will be obvious.

Fick's first law

Fick (1855) was the first person who found the analogy between transfer of heat by conduction with transfer of mass by diffusion both caused by random molecular motion. Just like Fourier's law for conduction of heat, Fick stated in his first law that the rate of transfer of diffusing substance through a unit area of a section of an isotropic substance is directly proportional to the concentration gradient measured normal to the section. Mathematically,

$$Q = -D \cdot \frac{\partial C}{\partial x} \quad 5.5.8$$

where

Q : mass transfer rate per unit area of section, kg moisture/s.m²

C : concentration of moisture, kg moisture/m³ sample

D : diffusion coefficient, m²/s

Fick's second law

The change of mass transfer rate with respect to space variable is proportional to the rate of change of concentration with respect to time. This is expressed by the following equation,

$$-\frac{\partial}{\partial x}(Q.A) = A.\frac{\partial C}{\partial t} \quad 5.5.9$$

For a slab geometry, the area terms A on both sides of Eq. (5.5.9) are canceled. It is not done since our interest is to develop a general procedure for any regular geometry, such as slab, cylindrical, or spherical. For the same purpose, from here on, the space variable x is substituted by r .

Both the sides of Eqs. (5.5.8) and (5.5.9) are divided by the density of the solid, ρ , to obtain

$$\frac{Q}{\rho} = -D.\frac{\partial M}{\partial r} \quad 5.5.10$$

and

$$-\frac{1}{\rho}.\frac{\partial}{\partial r}(Q.A) = A.\frac{\partial M}{\partial t} \quad 5.5.11$$

where

$$M : \text{product moisture, fraction db} = \frac{C}{\rho}.$$

Now transferring the value of Q from Eq. (5.5.10) to Eq. (5.5.11),

$$A.\frac{\partial M}{\partial t} = D.\frac{\partial}{\partial r}\left(A.\frac{\partial M}{\partial r}\right) \quad 5.5.12$$

The expression given by Eq. (5.5.12) states Fick's second law and is very similar to the heat transfer equation derived in Chapter 3.7. The only difference is that, in the case of mass transfer, the term $[\rho.C_p]$ does not appear in the left hand side of Eq.(5.5.12). In fact, in heat flow, the diffusing substance is heat and not temperature. Thus the factor $[\rho.C_p]$ is needed to convert temperature to the amount of heat per unit volume. In mass transfer, on the other hand, the diffusing substance is moisture itself, thus not requiring this factor.

In mass transfer, the diffusion coefficient is analogous to both thermal diffusivity and thermal conductivity. In drying, a single grain is considered exposed to heated air on all its sides and thus according to the convention used in Chapter 3.6, its side 1, that is, its outside surface, is considered having a convective boundary condition (Robbins condition), whereas due to symmetry, side 2, that is, its center, should have the derivative boundary

condition (Neumann condition). The initial and boundary conditions for this particular case are expressed below:

Initial condition

$$M(t = 0, r) = M_0 \quad 5.5.13$$

Boundary conditions

At side 1 (surface), the moisture diffused from node 2 to node 1 is transferred from node 1 to the atmosphere. Mathematically,

$$-D \cdot A_{1+1/2} \cdot \frac{\partial M}{\partial r} = h_m \cdot A_1 (M_{i,1} - M_e) \quad 5.5.14$$

where

$$\frac{\partial M}{\partial r} = \frac{M_{i,1} - M_{i,2}}{\Delta r} \quad \text{and} \quad A_{1+1/2} = \frac{A_1 + A_2}{2}$$

$$h_m : \text{convective mass transfer coefficient} = \frac{h}{\rho \cdot C_{pa}}, \text{ m/s.}$$

From Eq. (5.5.14), $M_{i,1}$ is expressed as

$$M_{i,1} = \frac{M_e + DD1 \cdot M_{i,2}}{1 + DD1}$$

where

$$DD1 = \frac{D(A_1 + A_2)}{2 \cdot A_1 \cdot h_m \cdot \Delta r}$$

At side 2 (center), due to symmetry, the Neumann condition exists , that is,

$$M_{i,n+1} = M_{i,n}$$

In heat transfer, the equations representing this situation are given in matrix form for the implicit method by Eq. (3.7.64). For mass transfer, the same representation is again shown here with appropriate notations:

$$\begin{bmatrix} (C_2 + CC_0) & C_3 & & & \\ C_1 & C_2 & C_3 & & \\ & C_1 & C_2 & C_3 & \\ \dots & \dots & \dots & \dots & \dots \\ & & C_1 & C_2 & C_3 \\ & & & C_1 & (C_2 + C_3) \end{bmatrix} \begin{bmatrix} M_{i+1,2} \\ M_{i+1,3} \\ M_{i+1,4} \\ \dots \\ M_{i+1,n-1} \\ M_{i+1,n} \end{bmatrix} = \begin{bmatrix} DD.M_{i,2} - CC \\ DD.M_{i,3} \\ DD.M_{i,4} \\ \dots \\ DD.M_{i,n-1} \\ DD.M_{i,n} \end{bmatrix} \quad 5.5.15$$

where C_1 , C_2 , C_3 , and DD are identical to the expressions given in relation to Eq. (3.7.45) in Chapter 3.7 and

$$CC = \frac{M_e.C_1}{1 + DD1} \quad CC_0 = \frac{DD1.C_1}{1 + DD1}$$

It should be pointed out here that the moisture contents calculated by the matrix (5.5.15) are actually nodal values, whereas the ones in Eqs. (5.5.1) through (5.5.3) are the average values. Thus at each time step it is required that all the nodal moisture contents be averaged by the expression (3.7.34), reexpressed as follows:

$$M = \frac{\sum_{j=1}^n v_j \cdot M_{i,j+1/2}}{V}$$

where

$$M_{i,j+1/2} = \frac{M_{i,j+1} + M_{i,j}}{2}$$

and

M : average product moisture, fraction db

V : total volume, m^3

v_j : volume of j -th segment, m^3

There is no way to control the time step in this case. It is rather calculated by the user-supplied values of the bed depth and the number of thin layers the bed is divided into. The step size along the bed depth, Δx , is calculated by dividing the bed depth by the number of divisions, and the time step is then calculated by dividing this step size by grain velocity. This is the reason why the diffusion equation for the moisture gradient has rarely been used in grain drying simulations. The implicit scheme overcomes this limitation due to its unconditional stability and is the reason for its use here.

At each time step, the program calculates the moisture gradient either by Eq. (5.5.6) when the thin-layer equation is chosen or by dividing the difference between average moistures calculated using Eq. (5.5.15) at two consecutive time steps by Δx , when the diffusion equation is chosen. In the

latter case, it calls the subroutine **COEFF** listed in Appendix A.5.6 and then the subroutine **TRIDGL** listed in Appendix A.3.7.3.

Though the program **DRY2000.FOR** is developed for simulating drying of any product by a concurrent flow dryer, the application problem to be solved is a particular case of drying parboiled rough rice, the configuration of which is approximated by an infinite cylinder. The coefficients of the thin-layer equation as well as the expression for diffusion coefficient are available (Chandra and Singh, 1984). Some expressions are given here that are applicable when parboiled rough rice is the product to be dried. The program segment **INCLUDE.FOR** listed in Appendix A.5.9 has the thin-layer subroutine and other functions relevant to rough rice. This part is to be written by the user if the simulation is to be made for some product other than parboiled rough rice.

Coefficients of the thin-layer drying equation

For air temperature within 100°C,

$$P = e^{a+b.R_h+c.\ln(1.8.T_a+32)+d.M_0}$$

and

$$Q = a_1 + b_1.R_h + c_1.T_a + d_1.M_0$$

where

M_0 : initial product moisture, fraction db

$a = -12.382707$; $b = 0.304496$; $c = 1.882463$; $d = 1.173742$

$a_1 = -0.251444$; $b_1 = 2.225801$; $c_1 = 2.719537 \times 10^{-3}$; $d_1 = 0.99337$

For an air temperature range of 100 – 158°C,

$$P = a + b.R_h + c.T_a + d.M_0$$

and

$$Q = a_1 + b_1.R_h + c_1.T_a + d_1.M_0$$

where

$a = -0.1700425$; $b = -2.173133 \times 10^{-2}$; $c = 3.145021 \times 10^{-3}$; $d = 0.149104$

$a_1 = 0.589744$; $b_1 = 0.190314$; $c_1 = 1.2106 \times 10^{-4}$; $d_1 = 7.833534 \times 10^{-2}$

Diffusion coefficient, D

[Function **FDIFF(T)** in Appendix A.5.9]

For a temperature range of 38 – 158°C,

$$D = 4.15744444 \times 10^{-6} \cdot e^{-3748.6/(T_a+273.15)}$$

Latent heat of evaporation, h_{fg}

[Function **FHFG**(TG,XMG) in Appendix A.5.9]

For a temperature range of 40 – 60°C,

$$h_{fg} = 1.862453 \times 10^6 - 1757.5 \cdot T_g \cdot M^{-0.21304}$$

Equilibrium moisture content, M_e

[Function **FXME**(T,RH) in Appendix A.5.9]

In percent dry basis, for a temperature range of 19 – 60°C,

$$M_e = 4.51 + 0.069 \cdot R_{hp} + 8.837 \cdot \sqrt{R_{hp}} - 0.027(T_g + 273.15) \sqrt{R_{hp}}$$

where

R_{hp} : relative humidity, percent.

Heat transfer coefficient, h

[Function **FH**(GPA) in Appendix A.5.9]

It is expressed as a function of dry air flow rate,

$$h = 24.19 \cdot G_a^{0.37}$$

Specific product surface area, a_s

This is the ratio of the heat transfer area with respect to grains and the total volume of grain bed area and is applicable when granular product is dried in bulk. Thus

$$a_s = \frac{\text{m}^2 \text{ of heat transfer area}}{\text{m}^3 \text{ of bed}}$$

For cylindrical geometry, if there are **n** number of grains of radius **R** in the bed, the total volume of bed including pore spaces is **V**, and the bed porosity is **ε**,

$$\begin{aligned} \text{Total volume of solids (grains)} &= V(1 - \epsilon) \\ &= n \cdot \pi \cdot R^2 \cdot l \end{aligned}$$

$$\text{Maximum surface area of solids} = n \cdot 2 \cdot \pi \cdot R \cdot l$$

$$\text{giving, } V = \frac{n \cdot \pi \cdot R^2 \cdot l}{1 - \epsilon} \quad \text{and}$$

$$a_s = \frac{n \cdot 2 \cdot \pi \cdot R \cdot l}{\frac{n \cdot \pi \cdot R^2 \cdot l}{1 - \epsilon}} = \frac{2(1 - \epsilon)}{R}$$

For spherical geometry, in a similar way, a_s is obtained as

$$a_s = \frac{3(1 - \epsilon)}{R}$$

Input data

Atmospheric pressure,	P = 101325 Pa
Initial dry bulb temperature of air,	T ₀ = 25 °C
Wet bulb temperature of air,	T _w = 18 °C
Heated air temperature,	T _a = 60 °C
Initial grain temperature,	T _g = 35 °C
Initial product moisture,	M ₀ = 50 % db
Final product moisture, (stopping criterion)	M _f = 20 % db
Flow rate of heated air,	A _{ve} l = 1 m ³ /s
Grain flow rate,	G _{ve} l = 0.2 kg/s
Bed cross-sectional area,	A = 0.5 m ²
Bed depth,	X _{bed} = 30 cm
Number of thin layers,	N = 20
Specific heat of the dry solid product,	C _{pp} = 1674.7 J/kg.°C
Bulk density of the dry product,	ρ _p = 640 kg/m ³
Porosity of the bulk product,	ε = 0.37
Product radius,	r _p = 1.135 mm
Number of nodes, (only for diffusion)	n = 50

With the above inputs, the program **DRY2000.FOR** produces the following outputs shown in Tables 5.5.1 and 5.5.2. It is observed that the use of the thin-layer equation estimates a slower drying rate than that by the diffusion equation. Also, the air and grain temperatures estimated by the former approach is on the higher side than that by the latter. In both the cases, fluid properties have been calculated at the arithmetic mean temperature of the air and the product.

Table 5.5.1. Drying simulation using thin-layer concept.

DEPTH [cm]	MOISTURE [% db]	HUMIDITY [fraction]	AIR TEMP. [°C]	GRAIN TEMP [°C]	TIME [s]
0.00	50.00	0.01004	60.00	35.00	0.00
3.00	48.44	0.01125	54.58	45.36	48.00
6.00	47.18	0.01223	52.14	46.95	96.00
9.00	46.20	0.01299	50.59	46.93	144.00
12.00	45.41	0.01360	49.44	46.55	192.00
15.00	44.74	0.01412	48.50	46.08	240.00
18.00	44.16	0.01457	47.70	45.62	288.00
21.00	43.65	0.01497	47.00	45.17	336.00
24.00	43.20	0.01532	46.38	44.75	384.00
27.00	42.79	0.01564	45.83	44.35	432.00
30.00	42.41	0.01593	45.33	43.98	480.00

Table 5.5.2. Drying simulation using diffusion concept.

DEPTH [cm]	MOISTURE [% db]	HUMIDITY [fraction]	AIR TEMP. [°C]	GRAIN TEMP [°C]	TIME [s]
0.00	50.00	0.01004	60.00	35.00	0.00
3.00	46.66	0.01263	53.13	40.42	48.00
6.00	45.24	0.01373	49.96	43.68	96.00
9.00	44.18	0.01456	48.17	44.04	144.00
12.00	43.32	0.01523	46.88	43.66	192.00
15.00	42.57	0.01580	45.83	43.12	240.00
18.00	41.92	0.01631	44.93	42.54	288.00
21.00	41.33	0.01677	44.12	41.97	336.00
24.00	40.79	0.01718	43.39	41.43	384.00
27.00	40.30	0.01756	42.72	40.91	432.00
30.00	39.85	0.01792	42.10	40.42	480.00

The successful use of the diffusion approach shown here will be extremely significant in the area of grain drying simulation. The program developed can be easily modified to get it to estimate the diffusion coefficient by iterative convergence procedure if reliable field data on drying of any biological product are available and a not-too-far-off initial guess can be provided. Since the diffusion coefficient is found to be an exponential function of the inverse of the absolute temperature of drying, experimental drying data using only two temperatures will suffice. The values of the diffusion coefficient of many biological products are available in the literature, which can be used as excellent initial guesses. On the other hand,

the thin-layer approach strictly requires thin-layer equations, which can only be developed by elaborate and controlled laboratory experiments.

References

1. **Chandra, P. K.**, Thin-layer and Concurrent-flow Rotary Drying Models for Parboiled Rice. Ph.D. dissertation, University of California, Davis, Microfiche # LD 781.D5j 1983 C431, 1983.
2. **Chandra, P. K. and Payne, F. A.**, Simple Marthematical Model Predicts Particle Separation by Cyclones. *Proceedings of the Third World Filtration Congress IV*, Ostend, Belgium, 1986.
3. **Chandra, P. K. and Singh, R. P.**, Thin-layer Drying of Parboiled Rice at Elevated Temperatures. *J. Food Sci.* 49(3): 905–909, 1984.
4. **Crank, J.**, *The Mathematics of Diffusion*. Oxford Science Publications, Clarendon Press, Oxford, UK, 414pp., 1975.
5. **Das, H. and Srivastav, P. P.**, Surface Heat Transfer Coefficient of Rice Puffed with Sand. *J. Food. Sci. Technol.* 26(1): 26–28, 1989.
6. **Fick, A.**, *Annln. Physics.* 170, 59, 1855.
7. **Keenan, J. H., Keyes, F. G., Hill, P. G., and Moore, J. G.**, *Steam Tables: Thermodynamic Properties of Water, Including Vapor, Liquid and Solid Phases (English Units)*. John Wiley & Sons, New York, 1969.
8. **Page, G. E.**, Factors Influencing the Maximum Rates of Air-drying Shelled Corn in Thin Layers. M.S dissertation, Purdue University, Lafayette, IN, 1949.
9. **Strauss, W.** *Industrial Gas Cleaning*. Edited by: P.V. Danckwerts. Pergamon Press, New York, Vol. 8, 160–213, 1974.

APPENDIX A.3.1. Program listings for numerical differentiation.

A.3.1.1. Subroutine **FINITE** for differentiation.

Program name : **FINITE.FOR**

```
C
C
C THIS SUBROUTINE CALCULATED FIRST, SECOND, THIRD AND FOURTH
C DERIVATIVE OF AN ANALYTIC FUNCTION USING NORMAL FORWARD,
C CENTRAL, BACKWARD AND DIFFERENCE METHODS.
C
C
C SUBROUTINE FINITE(F,I)
COMMON /BLOK1/ X,DY(2),H,K,KK
DOUBLE PRECISION X,DY,H,X1,X2,X3,X4,X11,X12,X13,X14
DOUBLE PRECISION F
C
C X1 = X + H
C X2 = X + 2.0*H
C X3 = X + 3.0*H
C X4 = X + 4.0*H
C X11 = X - H
C X12 = X - 2.0*H
C X13 = X - 3.0*H
C X14 = X - 4.0*H
C
C IF(KK .EQ. 1) THEN
C IF(K .EQ. 1) THEN
C
C FIRST DERIVATIVE IS CALCULATED BY FORWARD DIFFERENCE
C
C DY(I) = (F(X1) - F(X))/H
C ELSEIF(K .EQ. 2) THEN
C
C FIRST DERIVATIVE IS CALCULATED BY BACKWARD DIFFERENCE
C
C DY(I) = (F(X) - F(X11))/H
C ELSEIF(K .EQ. 3) THEN
C
C FIRST DERIVATIVE IS CALCULATED BY CENTRAL DIFFERENCE
C
C DY(I) = (F(X1) - F(X11))/2.0/H
C ENDIF
C ELSEIF(KK .EQ. 2) THEN
C IF(K .EQ. 1) THEN
C
C SECOND DERIVATIVE IS CALCULATED BY FORWARD DIFFERENCE
C
C DY(I) = (F(X2) - 2.0*F(X1) + F(X))/H/H
C ELSEIF(K .EQ. 2) THEN
C
C SECOND DERIVATIVE IS CALCULATED BY BACKWARD DIFFERENCE
```

```

C
      DY(I) = (F(X) - 2.0*F(X11) + F(X12))/H/H
      ELSEIF(K .EQ. 3) THEN
C
C      SECOND DERIVATIVE IS CALCULATED BY CENTRAL DIFFERENCE
C
      DY(I) = (F(X1) - 2.0*F(X) + F(X11))/H/H
      ENDIF
      ELSEIF(KK .EQ. 3) THEN
      IF(K .EQ. 1) THEN
C
C      THIRD DERIVATIVE IS CALCULATED BY FORWARD DIFFERENCE
C
      DY(I) = (F(X3) - 3.0*F(X2) + 3.0*F(X1) - F(X))/H**3
      ELSEIF(K .EQ. 2) THEN
C
C      THIRD DERIVATIVE IS CALCULATED BY BACKWARD DIFFERENCE
C
      DY(I) = (F(X) - 3.0*F(X11) + 3.0*F(X12) - F(X13))/H**3
      ELSEIF(K .EQ. 3) THEN
C
C      THIRD DERIVATIVE IS CALCULATED BY CENTRAL DIFFERENCE
C
      DY(I) = (F(X2)-2.0*F(X1)+2.0*F(X11)-F(X12))/(2.0*H**3)
      ENDIF
      ELSEIF(KK .EQ. 4) THEN
      IF(K .EQ. 1) THEN
C
C      FOURTH DERIVATIVE IS CALCULATED BY FORWARD DIFFERENCE
C
      DY(I) = (F(X4)-4.0*F(X3)+6.0*F(X2)-4.0*F(X1)+F(X))/H**4
      ELSEIF(K .EQ. 2) THEN
C
C      FOURTH DERIVATIVE IS CALCULATED BY BACKWARD DIFFERENCE
C
      DY(I) = (F(X)-4.0*F(X11)+6.0*F(X12)-4.0*F(X13)+F(X14))/H**4
      ELSEIF(K .EQ. 3) THEN
C
C      FOURTH DERIVATIVE IS CALCULATED BY CENTRAL DIFFERENCE
C
      DY(I) = (F(X2)-4.0*F(X1)+6.0*F(X)-4.0*F(X11)+F(X12))/H**4
      ENDIF
      ENDIF
      RETURN
      END
C

```

A.3.1.2. Subroutine **FINIT** for higher-order differentiation.

Program name : **FINIT.FOR**

C _____

```

C
C THIS SUBROUTINE CALCULATED FIRST, SECOND, THIRD AND FOURTH
C DERIVATIVE OF AN ANALYTIC FUNCTION USING HIGHER-ORDER
C FORWARD, AND BACKWARD CENTRAL DIFFERENCE METHODS.
C
C
C SUBROUTINE FINIT(F,I)
COMMON /BLOK1/ X,DY(2),H,K,KK
DOUBLE PRECISION X,DY,H,X1,X2,X3,X4,X5,X11,X12,X13,X14,X15
DOUBLE PRECISION F

C
X1 = X + H
X2 = X + 2.0*H
X3 = X + 3.0*H
X4 = X + 4.0*H
X5 = X + 5.0*H
X11 = X - H
X12 = X - 2.0*H
X13 = X - 3.0*H
X14 = X - 4.0*H
X15 = X - 5.0*H

C
IF(KK .EQ. 1) THEN
  IF(K .EQ. 1) THEN
C
C FIRST DERIVATIVE IS CALCULATED BY FORWARD DIFFERENCE
C
    DY(I) = (-F(X2)+4.0*F(X1)-3.0*F(X))/2.0/H
    ELSEIF(K .EQ. 2) THEN
C
C FIRST DERIVATIVE IS CALCULATED BY BACKWARD DIFFERENCE
C
    DY(I) = (3.0*F(X)-4.0*F(X11)+F(X12))/2.0/H
    ELSEIF(K .EQ. 3) THEN
C
C FIRST DERIVATIVE IS CALCULATED BY CENTRAL DIFFERENCE
C
    DY(I) = (-F(X2)+8.0*F(X1)-8.0*F(X11)+F(X12))/12.0/H
    ENDIF
    ELSEIF(KK .EQ. 2) THEN
      IF(K .EQ. 1) THEN
C
C SECOND DERIVATIVE IS CALCULATED BY FORWARD DIFFERENCE
C
        DY(I) = (-F(X3)+4.0*F(X2)-5.0*F(X1)+2.0*F(X))/H/H
        ELSEIF(K .EQ. 2) THEN
C
C SECOND DERIVATIVE IS CALCULATED BY BACKWARD DIFFERENCE
C
        DY(I) = (2.0*F(X)-5.0*F(X11)+4.0*F(X12)-F(X13))/H/H
        ELSEIF(K .EQ. 3) THEN
C
C SECOND DERIVATIVE IS CALCULATED BY CENTRAL DIFFERENCE

```

```

C
      DY(I) = (-F(X2)+16.0*F(X1)-30.0*F(X)+16.0*F(X11)-F(X12))/
1      (12.0*H*H)
      ENDIF
      ELSEIF(KK .EQ. 3) THEN
      IF(K .EQ. 1) THEN
C
C   THIRD DERIVATIVE IS CALCULATED BY FORWARD DIFFERENCE
C
      DY(I) = (-3.0*F(X4)+14.0*F(X3)-24.0*F(X2)+18.0*F(X1)-5.0*
1      F(X))/(2.0*H**3)
      ELSEIF(K .EQ. 2) THEN
C
C   THIRD DERIVATIVE IS CALCULATED BY BACKWARD DIFFERENCE
C
      DY(I) = (5.0*F(X)-18.0*F(X11)+24.0*F(X12)-14.0*F(X13)+3.0*
1      F(X14))/(2.0*H**3)
      ELSEIF(K .EQ. 3) THEN
C
C   THIRD DERIVATIVE IS CALCULATED BY CENTRAL DIFFERENCE
C
      DY(I) = (-F(X3)+8.0*F(X2)-13.0*F(X1)+13.0*F(X11)-8.0*F(X12)+
1      F(X13))/(8.0*H**3)
      ENDIF
      ELSEIF(KK .EQ. 4) THEN
      IF(K .EQ. 1) THEN
C
C   FOURTH DERIVATIVE IS CALCULATED BY FORWARD DIFFERENCE
C
      DY(I) = (-2.0*F(X5)+11.0*F(X4)-24.0*F(X3)+26.0*F(X2)-14.0*
1      F(X1)+3.0*F(X))/H**4
      ELSEIF(K .EQ. 2) THEN
C
C   FOURTH DERIVATIVE IS CALCULATED BY BACKWARD DIFFERENCE
C
      DY(I) = (3.0*F(X)-14.0*F(X11)+26.0*F(X12)-24.0*F(X13)+11.0*
1      F(X14)-2.0*F(X15))/H**4
      ELSEIF(K .EQ. 3) THEN
C
C   FOURTH DERIVATIVE IS CALCULATED BY CENTRAL DIFFERENCE
C
      DY(I) = (-F(X3)+12.0*F(X2)-39.0*F(X1)+56.0*F(X)-39.0*F(X11)+
1      12.0*F(X12)-F(X13))/(6.0*H**4)
      ENDIF
      ENDIF
      RETURN
      END
C

```

Program name : **FCTN.FOR**

```

C
C
C
C THIS FUNCTION DEFINES THE FIRST-ORDER DIFFERENTIAL EQUATION
C
C
C F = X*DSIN(X)
C RETURN
C END
C

```

A.3.1.4. Main program for differentiation by numerical methods.

Program name: **MAIN.FOR**

```

C
C
C THIS PROGRAM COUPLED WITH THE SUBROUTINE 'FINITE' CAN
C DIFFERENTIATE ANY FUNCTION TO GIVE FIRST, SECOND, THIRD AND
C FOURTH DERIVATIVES USING NORMAL AND HIGHER-ORDER METHODS.
C
C Program Developed by: Prof. Prabir K. Chandra
C
C
C COMMON /BLOK1/ X,DY(2),H,K,KK
C CHARACTER S *32, S1 *19, S2 *19
C DOUBLE PRECISION X,DY,H
C DOUBLE PRECISION F
C EXTERNAL F
C
C OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C KK : INDEX NUMBER [1] for Y'; [2] for Y"; [3] for Y''';
C [4] for d^4Y/dY^4
C K : INDEX NUMBER [1] for FORWARD; [2] for BACKWARD;
C [3] for CENTRAL DIFFERENCE SCHEME
C X,H : VALUE OF INDEPENDENT VARIABLE AND STEP SIZE.
C
C DO 10 II=1,100
C WRITE(*,20)
C READ(*,*) KK
C IF(KK .EQ. 1) THEN
C S = 'FIRST DERIVATIVE'
C ELSEIF(KK .EQ. 2) THEN
C S = 'SECOND DERIVATIVE'
C ELSEIF(KK .EQ. 3) THEN
C S = 'THIRD DERIVATIVE'
C ELSEIF(KK .EQ. 4) THEN
C S = 'FOURTH DERIVATIVE'
C ELSE

```



```

        CLOSE(20,STATUS='KEEP')
        STOP
    ENDIF
    WRITE(*,30)
    READ(*,*) K
    IF(K .EQ. 1) THEN
        S1 = 'FORWARD DIFFERENCE'
    ELSEIF(K .EQ. 2) THEN
        S1 = 'BACKWARD DIFFERENCE'
    ELSEIF(K .EQ. 3) THEN
        S1 = 'CENTRAL DIFFERENCE'
    ELSE
        GOTO 10
    ENDIF
    WRITE(*,40)
    READ(*,*) L
    WRITE(*,50)
    READ(*,*) X,H
C
C NUMERICAL METHOD BEGINS BY CALLING SUBROUTINE 'FINITE' or 'FINIT'
C RICHARDSON'S EXTRAPOLATION IS USED FOR REFINEMENT ONLY IN CASE
C OF HIGHER-THAN-FIRST-ORDER METHODS ...
C
    IF(L .EQ. 1) THEN
        S2 = 'Normal-ORDER method'
        CALL FINITE(F,1)
    ELSE
        S2 = 'Higher-ORDER method'
        CALL FINIT(F,1)
    ENDIF
    IF(((K .EQ. 3) .AND. (L .EQ. 1)) .OR. (L .EQ. 2)) THEN
        H = H/2.0
        IF(L .EQ. 1) THEN
            CALL FINITE(F,2)
            J = 2
        ELSE
            CALL FINIT(F,2)
        ENDIF
        IF(L .EQ. 2) THEN
            IF(K .LT. 3) THEN
                J = 2
            ELSE
                J = 4
            ENDIF
        ENDIF
        DY(1) = DY(2) + (DY(2)-DY(1))/(2.0**J-1.0)
        H = 2.0*H
    ENDIF
C
    WRITE(*,60) S,S1,S2
    WRITE(20,70) S,S1,S2
    WRITE(*,80) DY(1),X,H
    WRITE(20,80) DY(1),X,H

```

```

        PAUSE
10 CONTINUE
C
C ***** FORMAT STATEMENTS *****
C
20 FORMAT(15(/)1X,79('#)/1X,'#',77X,'#/1X,'#',26X,'Numerical ',
1      'DIFFERENTIATION',26X,'#/1X,'#',77X,'#/1X,'#',30X,
1      'MAIN OPTION MENU',31X,'#/1X,'#',77X,'#/1X,'#',25X,
1      'CHOOSE ANY OPTION BY NUMBER',25X,'#/1X,'#',25X,
1      27('-'),25X,'#/1X,'#',77X,'#/1X,'#',28X,'1: FIRST ',
1      'DERIVATIVE',30X,'#/1X,'#',28X,'2: SECOND DERIVATIVE',
1      29X,'#/1X,'#',28X,'3: THIRD DERIVATIVE',30X,'#/1X,
1      '#',28X,'4: FOURTH DERIVATIVE',29X,'#/1X,'#',77X,'#/
1      1X,'#',23X,'[Type any OTHER number to QUIT]',23X,'#/
1      1X,'#',77X,'#/1X,79('#)//)
30 FORMAT(15(/)1X,79('#)/1X,'#',77X,'#/1X,'#',26X,'Numerical ',
1      'DIFFERENTIATION',26X,'#/1X,'#',77X,'#/1X,'#',31X,
1      'SECONDARY MENU',32X,'#/1X,'#',77X,'#/1X,'#',23X,
1      'CHOOSE ANOTHER OPTION BY NUMBER',23X,'#/1X,'#',23X,
1      31('-'),23X,'#/1X,'#',77X,'#/1X,'#',28X,'1: FORWARD',
1      ' DIFFERENCE',28X,'#/1X,'#',28X,'2: BACKWARD DIFFER',
1      'ENCE',27X,'#/1X,'#',28X,'3: CENTRAL DIFFERENCE',28X,
1      '#/1X,'#',77X,'#/1X,'#',12X,'[Type any OTHER number',
1      ' to RETURN to MAIN OPTION MENU]',12X,'#/1X,'#',77X,
1      '#/1X,79('#)//)
40 FORMAT(25(/)27X,'Choose ORDER of the method',27X,26('-')/2X,
1      'Press [1] - FIRST-ORDER FORWARD and BACKWARD; SECOND-',
1      'ORDER CENTRAL scheme',2X,'Press [2] - SECOND-ORDER ',
1      'FORWARD and BACKWARD; FOURTH-ORDER CENTRAL
scheme//)
50 FORMAT(25(/)15X,'TYPE in the VALUE OF X and STEP SIZE and ',
1      '<ENTER>.'/15X,'[STEP size should be POSITIVE and LESS ',
1      'THAN ONE]//)
60 FORMAT(20(/)12X,A32,' - ',A19/12X,54('-')/29X,['.A19,']/)
70 FORMAT(/12X,A32,' - ',A19/12X,54('-')/29X,['.A19,']/)
80 FORMAT(/1X,'Required DERIVATIVE = ',D14.8,2X,'at X = ',F8.4,2X,
1      'with STEP size = ',F6.5)
      STOP
      END
C
C
C SUBROUTINES FINITE(F,I) and FINIT(F,I) ARE INCLUDED
C
C
$INCLUDE:'FINITE.FOR'
$INCLUDE:'FINIT.FOR'
C
C
C THIS FUNCTION DEFINES THE FIRST-ORDER DIFFERENTIAL EQUATION.
C
C
DOUBLE PRECISION FUNCTION F(X)
DOUBLE PRECISION X

```

C

\$INCLUDE:'FCTN.FOR'

C

APPENDIX A.3.2. Program listings for calculating root(s) of an algebraic equation.

A.3.2.1. Subroutine **SEARCH** for roughly tracing a real root.

Program name : **SEARCH.FOR**

```
C _____
C
C THIS SUBROUTINE MAKES A ROUGH SEARCH FOR THE ROOTS
C _____
C
  SUBROUTINE SEARCH(F)
  COMMON /BLOK1/ X,X1,X2,ERROR,E,KK,K
  COMMON /BLOK2/ X0,X00,H,A,B,C,NR,I2
  CHARACTER *1 S
C
  IF(NR .EQ. 1) THEN
    IF(X0 .GT. X00) THEN
      TEMP = X00
      X00 = X0
      X0 = TEMP
    ENDIF
    X = X0
    X1 = X0
    I2 = ABS(X00 - X0)/H + 1
    DO 10 I=1,I2
      K = I
      X = X + H
      A = F(X1)
      B = F(X)
      IF(ABS(A) .LE. ERROR) THEN
        X = X1
        X2 = F(X)
        RETURN
      ELSEIF(ABS(B) .LE. ERROR) THEN
        X2 = F(X)
        RETURN
      ENDIF
      C = A*B
      IF(C .GT. 1.0E37) THEN
        C = 1.0
        K = I2
        RETURN
      ENDIF
      IF(C .LT. 0.0) RETURN
      X1 = X
10  CONTINUE
    RETURN
  END
```

C _____

A.3.2.2. Subroutine **ROOT** for calculating real roots by five different methods.

Program name : **ROOT.FOR**

```

C _____
C
C THIS SUBROUTINE EMPLOYS (I) NEWTON-RAPHSON, (II) MODIFIED NEWTON,
C (III) SECANT (IV) BISECTION AND (V) FALSE POSITION METHODS.
C _____
C
C SUBROUTINE ROOT(F,FD,FDD)
C COMMON /BLOK1/ X,X1,X2,ERROR,E,KK,K
C
C E = 1.0
C IF(KK .EQ. 1) THEN
C NEWTON-RAPHSON METHOD BEGINS ...
C
C DO 10 I=1,1000
C K = I
C E = FD(X)
C IF(E .EQ. 0.0) RETURN
C D = -(F(X)/E)
C X = X + D
C IF(ABS(D) .LE. ERROR) THEN
C X2 = F(X)
C RETURN
C ENDIF
10 CONTINUE
C ELSEIF(KK .EQ. 2) THEN
C MODIFIED NEWTON METHOD BEGINS ...
C
C DO 20 I=1,1000
C K = I
C A = F(X)
C E = FD(X)
C IF(E .EQ. 0.0) RETURN
C U = A/E
C UD = 1.0 - A*FDD(X)/E/E
C D = -U/UD
C X = X + D
C IF(ABS(D) .LE. ERROR) THEN
C X2 = F(X)
C RETURN
C ENDIF
20 CONTINUE
C ELSEIF(KK .EQ. 3) THEN
C SECANT METHOD STARTS ...

```

C

```

D = X - X1
FOLD = F(X1)
DO 30 I=1,1000
    K = I
    FNEW = F(X)
    E = FNEW - FOLD
    IF(E .EQ. 0.0) RETURN
    D = -FNEW*D/E
    X = X + D
    IF((ABS(D) .LE. ERROR) .OR. (ABS(E) .LE. ERROR)) THEN
        X2 = F(X)
        RETURN
    ENDIF
    FOLD = FNEW

```

```

30  CONTINUE
    ELSEIF(KK .EQ. 4) THEN

```

C

C BISECTION METHOD BEGINS ...

C

```

A = X1
B = X
DO 40 I=1,1000
    K = I
    XM = (A + B)/2.0
    X2 = F(XM)
    C = F(B)*X2
    IF(C .LT. 0.0) THEN
        A = XM
    ELSEIF(C .GT. ERROR) THEN
        B = A
        A = XM
    ELSEIF((C .LE. ERROR) .OR. (ABS(X2) .LE. ERROR)) THEN
        X = XM
        RETURN
    ENDIF
40  CONTINUE
    ELSEIF(KK .EQ. 5) THEN

```

C

C FALSE POSITION METHOD BEGINS ...

C

```

A = X1
B = X
DO 50 I=1,1000
    K = I
    FA = F(A)
    FB = F(B)
    E = FB - FA
    IF(E .EQ. 0.0) RETURN
    X = (A*FB - B*FA)/E
    FX = F(X)
    C = FA*FX
    IF(C .LT. 0.0) THEN

```

```

      B = X
      FA = FA/2.0
      ELSEIF(C .GT. ERROR) THEN
        A = X
        FB = FB/2.0
      ELSEIF((C .LE. ERROR) .OR. (ABS(B-A) .LE. ERROR)) THEN
        X2 = FX
        RETURN
      ENDIF
50 CONTINUE
      ENDIF
      RETURN
      END

```

C

A.3.2.3. Functions **F**, **FD** and **FDD** for representing the function, and its first and second derivatives.

Program name : **FCTN.FOR**

```

C
C
C  THESE THREE FUNCTIONS ARE TO BE SUPPLIED BY THE USER ...
C      Program name : FCTN.FOR
C

```

```

C
C  FUNCTION F(X)
C
C      F = X**6+X**5-49.0*X**4+69.0*X**3+120.0*X**2+98.0*X-240.0
      RETURN
      END

```

```

C
C
C  THIS FUNCTION DEFINES THE FIRST DERIVATIVE ...
C

```

```

C
C  FUNCTION FD(X)
C
C      FD = 6.0*X**5+5.0*X**4-196.0*X**3+207.0*X**2+240.0*X+98.0
      RETURN
      END

```

```

C
C
C  THIS FUNCTION DEFINES THE SECOND DERIVATIVE ...
C

```

```

C
C  FUNCTION FDD(X)
C
C      FDD = 30.0*X**4+20.0*X**3-588.0*X**2+414.0*X+240.0
      RETURN

```

END

A.3.2.4. Main program **MAIN** for calculating a real root one at a time calling subroutines **SEARCH** and **ROOT**.

Program name : **MAIN.FOR**

```

C _____
C
C THIS PROGRAM COUPLED WITH THE SUBROUTINE 'ROOT' CAN SOLVE ANY
C LINEAR ALGEBRAIC EQUATION EMPLOYING (I) NEWTON-RAPHSON
C (II) MODIFIED NEWTON (III) SECANT AND (VI) BISECTION METHODS.
C
C
C Program Developed by: Prof. Prabir K. Chandra
C _____
C
COMMON /BLOK1/ X,X1,X2,ERROR,E,KK,K
COMMON /BLOK2/ X0,X00,H,A,B,C,NR,I2
CHARACTER S *22, S1 *1
EXTERNAL F,FD,FDD
C
C OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C KK : INDEX NUMBER [1] NEWTON-RAPHSON; [2] MODIFIED
C NEWTON; [3] SECANT; [4] BISECTION and [5] FALSE
C POSITION METHOD
C K1, K2 : X-INTERVAL FOR SEARCHING ROOTS
C X, ERROR : THE ROOT AND ERROR LIMIT FOR CONVERGENCE
C K : NUMBER OF ITERATIONS
C
DO 30 II=1,100
  WRITE(*,40)
  READ(*,*) KK
  IF(KK .EQ. 1) THEN
    S = 'NEWTON-RAPHSON METHOD'
  ELSEIF(KK .EQ. 2) THEN
    S = 'MODIFIED NEWTON METHOD'
  ELSEIF(KK .EQ. 3) THEN
    S = 'SECANT METHOD'
  ELSEIF(KK .EQ. 4) THEN
    S = 'BISECTION METHOD'
  ELSEIF(KK .EQ. 5) THEN
    S = 'FALSE POSITION METHOD'
  ELSE
    WRITE(20,140)
    CLOSE(20,STATUS='KEEP')
    STOP
  ENDIF
  WRITE(20,120) S
  WRITE(20,50)

```



```

      NR = 1
10    WRITE(*,60) S
      READ(*,*) K1,K2,H,ERROR
      X0 = FLOAT(K1)
      X00 = FLOAT(K2)

C
C  PRELIMINARY SEARCH IS MADE BY SUBROUTINE 'SEARCH'
C
      CALL SEARCH(F)
      IF(X0 .GT. X00) GOTO 30
      IF((K .EQ. I2) .AND. (C .GT. 0.0)) THEN
        WRITE(*,90) X0,X
        READ(*,100) S1
        IF((S1 .EQ. 'M') .OR. (S1 .EQ. 'm')) THEN
          IF(KK .GE. 4) THEN
            WRITE(*,150)
            PAUSE
            GOTO 30
          ENDIF
          GOTO 20
        ELSEIF((S1 .EQ. 'I') .OR. (S1 .EQ. 'i')) THEN
          GOTO 10
        ELSE
          GOTO 30
        ENDIF
      ENDIF
20    CONTINUE

C
C  NUMERICAL METHOD BEGINS BY CALLING SUBROUTINE 'ROOT'
C
      IF((ABS(A) .LE. ERROR) .OR. (ABS(B) .LE. ERROR)) THEN
        WRITE(*,70)
        WRITE(20,70)
        PAUSE
      ELSE
        CALL ROOT(F,FD,FDD)
        IF((E .EQ. 0.0) .OR. (K .EQ. 1000)) THEN
          WRITE(*,80) X0,X
          READ(*,100) S1
          IF((S1 .EQ. 'N') .OR. (S1 .EQ. 'n')) THEN
            GOTO 30
          ELSE
            GOTO 10
          ENDIF
        ENDIF
      ENDIF
      ENDIF

C
      WRITE(*,110) S
      WRITE(*,50)
      WRITE(*,130) NR,X,X2,K
      WRITE(20,130) NR,X,X2,K
      PAUSE
30  CONTINUE

```

```

C
C ***** FORMAT STATEMENTS *****
C
40 FORMAT(15(/)1X,79('#')/1X,'#',77X,'#/1X,'#',16X,'SOLUTION OF ',
1      'LINEAR ALGEBRAIC EQUATION (ROOT)',17X,'#/1X,'#',77X,
1      '#/1X,'#',30X,'MAIN OPTION MENU',31X,'#/1X,'#',77X,'#',
1      '/1X,'#',25X,'CHOOSE ANY METHOD BY NUMBER',25X,'#/1X,'#',
1      '25X,27('-')',25X,'#/1X,'#',77X,'#/1X,'#',30X,'1: NEWTON',
1      '-RAPHSON',30X,'#/1X,'#',30X,'2: MODIFIED NEWTON',29X,
1      '#/1X,'#',30X,'3: SECANT',38X,'#/1X,'#',30X,'4: BISE',
1      'CTION',35X,'#/1X,'#',30X,'5: FALSE POSITION',30X,'#/1X,
1      '#',77X,'#/1X,'#',23X,['Type any OTHER number to QUIT'],
1      '23X,'#/1X,'#',77X,'#/1X,79('#')//)
50 FORMAT(14X,'Root #   Root Value   Function Value   Iterations'
1      /14X,52('-'))
60 FORMAT(20(/)30X,A22//15X,50('_')//21X,'Give X-interval (TWO ',
1      'INTEGER values),/10X,'search SPACING and ERROR bound ',
1      '(TWO REAL values) and <ENTER>./18X,['ERROR bound is ',
1      'generally 1.0E-06 or LARGER'/15X,50('_')//)
70 FORMAT(5X,'Root below was found by rough SEARCH - Probably it ',
1      'is a MULTIPLE ROOT.')
80 FORMAT(20(/)2X,78('-')//2X,'There may be NO ROOT in this ',
1      'INTERVAL of ',1PE15.8,' and ',1PE15.8,'./21X,' - TRY ',
1      'another INTERVAL/SPACING (Y/N)?/2X,78('-')//)
90 FORMAT(20(/)2X,78('_')//2X,'A MULTIPLE or NO ROOT in this ',
1      'INTERVAL of ',1PE15.8,' and ',1PE15.8,'./22X,'CHOOSE ',
1      'ANY OF THE FOLLOWING OPTIONS'/23X,['I] : Change INTER',
1      'VAL/SPACING'/23X,['M] : Go ahead to find the ROOT'/
1      '23X,['N] : Go back to MAIN OPTION MENU'/2X,78('_')//)
100 FORMAT(A1)
110 FORMAT(20(/)30X,A22/30X,22('-')/)
120 FORMAT(/30X,A22/30X,22('-')/)
130 FORMAT(16X,I2,4X,1PE15.8,2X,1PE15.8,4X,I4)
140 FORMAT(14X,52('-'))
150 FORMAT(20(/)3X,'Function does NOT CHANGE SIGN ... BISECTION/FALSE',
1      ' POSITION does NOT work.'////)
      STOP
      END
C
C
C SUBROUTINE SEARCH(F) IS INCLUDED ...
C
C
C $INCLUDE:'SEARCH.FOR'
C
C
C SUBROUTINE ROOT(F,FD,FDD) IS INCLUDED ...
C
C
C $INCLUDE:'ROOT.FOR'
C
C
C FUNCTIONS F, FD AND FDD DEFINE ALGEBRAIC EQUATION, ITS

```

C FIRST AND SECOND DERIVATIVES RESPECTIVELY ...THESE ARE
C INCLUDED IN AN EXTERNAL FILE CALLED "FCTN.FOR".

C _____
C
C \$INCLUDE:'FCTN.FOR'
C _____

A.3.2.5. Main program **ZEROS** for calculating real roots within a given interval calling subroutines **SEARCH** and **ROOT**.

Program name : **ZEROS.FOR**

C _____
C
C THIS PROGRAM COUPLED WITH THE SUBROUTINE 'ROOT' CAN SOLVE ANY
C LINEAR ALGEBRAIC EQUATION EMPLOYING (I) NEWTON-RAPHSON
C (II) MODIFIED NEWTON (III) SECANT (IV) BISECTION AND (V) FALSE
C POSITION METHODS.

C
C Program Developed by: Prof. Prabir K. Chandra
C _____
C

C COMMON /BLOK1/ X,X1,X2,ERROR,E,KK,K
C COMMON /BLOK2/ X0,X00,H,A,B,C,NR,I2
C CHARACTER S *22, S1 *1
C EXTERNAL F,FD,FDD
C
C OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C KK : INDEX NUMBER [1] NEWTON-RAPHSON; [2] MODIFIED
C NEWTON; [3] SECANT; [4] BISECTION and [5] FALSE
C POSITION METHOD
C K1, K2 : X-INTERVAL FOR SEARCHING ROOTS
C X, ERROR : THE ROOT AND ERROR LIMIT FOR CONVERGENCE
C K : NUMBER OF ITERATIONS

C
C DO 40 II=1,100
C WRITE(*,50)
C READ(*,*) KK
C IF(KK .EQ. 1) THEN
C S = 'NEWTON-RAPHSON METHOD'
C ELSEIF(KK .EQ. 2) THEN
C S = 'MODIFIED NEWTON METHOD'
C ELSEIF(KK .EQ. 3) THEN
C S = 'SECANT METHOD'
C ELSEIF(KK .EQ. 4) THEN
C S = 'BISECTION METHOD'
C ELSEIF(KK .EQ. 5) THEN
C S = 'FALSE POSITION METHOD'
C ELSE

```

        WRITE(20,150)
        CLOSE(20,STATUS='KEEP')
        STOP
    ENDIF
    WRITE(20,130) S
    WRITE(20,60)
    NR = 1
10    WRITE(*,70) S
    READ(*,*) K1,K2,H,ERROR
    X0 = FLOAT(K1)
    X00 = FLOAT(K2)
C
C  PRELIMINARY SEARCH IS MADE BY SUBROUTINE 'SEARCH'
C
20    CALL SEARCH(F)
    IF(X0 .GT. X00) GOTO 40
    IF((K .EQ. I2) .AND. (C .GT. 0.0)) THEN
        WRITE(*,100) X0,X
        READ(*,110) S1
        IF((S1 .EQ. 'M') .OR. (S1 .EQ. 'm')) THEN
            IF(KK .GE. 4) THEN
                WRITE(*,160)
                PAUSE
                GOTO 40
            ENDIF
            GOTO 30
        ELSEIF((S1 .EQ. 'T') .OR. (S1 .EQ. 't')) THEN
            GOTO 10
        ELSE
            GOTO 40
        ENDIF
    ENDIF
30    CONTINUE
C
C  NUMERICAL METHOD BEGINS BY CALLING SUBROUTINE 'ROOT'
C
    IF((ABS(A) .LE. ERROR) .OR. (ABS(B) .LE. ERROR)) THEN
        WRITE(*,80)
        WRITE(20,80)
        PAUSE
    ELSE
        CALL ROOT(F,FD,FDD)
        IF((E .EQ. 0.0) .OR. (K .EQ. 1000)) THEN
            WRITE(*,90) X0,X
            READ(*,110) S1
            IF((S1 .EQ. 'N') .OR. (S1 .EQ. 'n')) THEN
                GOTO 40
            ELSE
                X0 = X
                GOTO 20
            ENDIF
        ENDIF
    ENDIF
ENDIF

```

C

```

IF(X00.GE. X0) THEN
  WRITE(*,120) S
  WRITE(*,60)
  WRITE(*,140) NR,X,X2,K
  WRITE(20,140) NR,X,X2,K
  PAUSE
  X0 = X + 0.0001
  NR = NR + 1
  GOTO 20
ENDIF

```

40 CONTINUE

C

C ***** FORMAT STATEMENTS *****

C

```

50 FORMAT(15(/)1X,79('#')/1X,'#',77X,'#/1X,'#',16X,'SOLUTION OF ',
1      'LINEAR ALGEBRAIC EQUATION (ROOT)',17X,'#/1X,'#',77X,
1      '#/1X,'#',30X,'MAIN OPTION MENU',31X,'#/1X,'#',77X,'#'
1      '/1X,'#',25X,'CHOOSE ANY METHOD BY NUMBER',25X,'#/1X,'#',
1      '25X,27('-'),25X,'#/1X,'#',77X,'#/1X,'#',30X,'1: NEWTON',
1      '-RAPHSON',30X,'#/1X,'#',30X,'2: MODIFIED NEWTON',29X,
1      '#/1X,'#',30X,'3: SECANT',38X,'#/1X,'#',30X,'4: BISE',
1      '3X,'#/1X,'#',77X,'#/1X,79('#')/)
60 FORMAT(14X,'Root #   Root Value   Function Value   Iterations'
1      '/14X,52('-'))
70 FORMAT(20(/)30X,A22//15X,50('_')//21X,'Give X-interval (TWO ',
1      'INTEGER values),/10X,'search SPACING and ERROR bound ',
1      '(TWO REAL values) and <ENTER>./18X,'[ERROR bound is ',
1      'generally 1.0E-06 or LARGER'/15X,50('_')/)
80 FORMAT(5X,'Root below was found by rough SEARCH - Probably it ',
1      'is a MULTIPLE ROOT.')
90 FORMAT(20(/)2X,78('_')//2X,'There may be NO ROOT in this ',
1      'INTERVAL of ',1PE15.8,' and ',1PE15.8,'./19X,' - TRY ',
1      'rest of the INTERVAL/SPACING (Y/N)?/2X,78('_')/)
100 FORMAT(20(/)2X,78('_')//2X,'A MULTIPLE or NO ROOT in this ',
1      'INTERVAL of ',1PE15.8,' and ',1PE15.8,'./22X,'CHOOSE ',
1      'ANY OF THE FOLLOWING OPTIONS//23X,'[I] : Change INTER',
1      'VAL/SPACING'/23X,'[M] : Go ahead to find the ROOT'/
1      '23X,'[N] : Go back to MAIN OPTION MENU'/2X,78('_')/)
110 FORMAT(A1)
120 FORMAT(20(/)30X,A22/30X,22('-')/)
130 FORMAT(/30X,A22/30X,22('-')/)
140 FORMAT(16X,I2,4X,1PE15.8,2X,1PE15.8,4X,I4)
150 FORMAT(14X,52('-'))
160 FORMAT(20(/)3X,'Function does NOT CHANGE SIGN ... BISECTION/FALSE',
1      ' POSITION does NOT work.////)
      STOP
      END

```

C

C

C SUBROUTINE SEARCH(F) IS INCLUDED ...

C

C

```
$INCLUDE:'SEARCH.FOR'
```

```
C
```

```
C
```

```
C  SUBROUTINE ROOT(F,FD,FDD) IS INCLUDED ...
```

```
C
```

```
C
```

```
$INCLUDE:'ROOT.FOR'
```

```
C
```

```
C
```

```
C  FUNCTIONS F, FD AND FDD DEFINE ALGEBRAIC EQUATION, ITS  
C  FIRST AND SECOND DERIVATIVES RESPECTIVELY ...THESE ARE  
C  INCLUDED IN AN EXTERNAL FILE CALLED "FCTN.FOR".
```

```
C
```

```
C
```

```
$INCLUDE:'FCTN.FOR'
```

```
C
```



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

APPENDIX A.3.3. Program listings for numerical integration.

A.3.3.1. Subroutine **WEIGHT** to read zeros and weights of Gauss
Quadrature method.

Program name: **WEIGHT.FOR**

```
C _____
C
C THIS SUBROUTINE READS ZEROS AND WEIGHTS OF GAUSS-QUADRATURE
C METHOD AND SUPPLIES THEM TO THE MAIN PROGRAM WHEN CALLED.
C _____
C
C SUBROUTINE WEIGHT
C COMMON /BLOK/ Z(24,24),W(24,24),T(15,15),A,B,XI,H,ER,K,KK
C
C THE DATA FILE "GAUSS.DAT" CONTAINS DATA FOR GAUSS-LEGENDRE
C QUADRATURE METHOD. THIS FILE SHOULD EXIST WITH DATA.
C
C OPEN(10,FILE='GAUSS.DAT',STATUS='OLD')
C
C ZEROS AND WEIGHT FACTORS ARE READ FROM 'GAUSS.DAT'
C DATA FOR EVEN NUMBER OF PANELS ARE READ ...
C
C DO 10 I=2,12,2
C READ(10,*) (Z(I,J),W(I,J),J=1,I/2)
10 CONTINUE
C
C DATA FOR ODD NUMBER OF PANELS ARE READ ...
C
C DO 20 I=3,9,2
C READ(10,*) (Z(I,J),W(I,J),J=1,I/2+1)
20 CONTINUE
C
C OTHER ODD DATA FOR PANELS 15,16,20 AND 24 ARE READ ...
C
C READ(10,*) (Z(15,J),W(15,J),J=1,8)
C READ(10,*) (Z(16,J),W(16,J),J=1,8)
C READ(10,*) (Z(20,J),W(20,J),J=1,10)
C READ(10,*) (Z(24,J),W(24,J),J=1,12)
C
C REWIND 10
C RETURN
C END
C _____
```


A.3.3.2. Data file **GAUSS.DAT** of zeros and weights of Gauss Quadrature.

```

0.577350269189626,1.000000000000000
0.339981043584856,0.652145154862546,0.861136311594053,0.347854845137454
0.238619186083197,0.467913934572691,0.661209386466265,0.360761573048139
0.932469514203152,0.171324492379170
0.183434642495650,0.362683783378362,0.525532409916329,0.313706645877887
0.796666477413627,0.222381034453374,0.960289856497536,0.101228536290376
0.148874338981631,0.295524224714753,0.433395394129247,0.269266719309996
0.679409568299024,0.219086362515982,0.865063366688985,0.149451349150581
0.973906528517172,0.066671344308688
0.125233408511469,0.249147045813403,0.367831498998180,0.233492536538355
0.587317954286617,0.203167426723066,0.769902674194305,0.160078328543346
0.904117256370475,0.106939325995318,0.981560634246719,0.047175336386512
0.000000000000000,0.888888888888889,0.774596669241483,0.555555555555556
0.000000000000000,0.568888888888889,0.538469310105683,0.478628670499366
0.906179845938664,0.236926885056189
0.000000000000000,0.417959183673469,0.405845151377397,0.381830050505119
0.741531185599394,0.279705391489277,0.949107912342759,0.129484966168870
0.000000000000000,0.330239355001260,0.324253423403809,0.312347077040003
0.613371432700590,0.260610696402935,0.836031107326636,0.180648160694857
0.968160239507626,0.081274388361574
0.000000000000000,0.202578241925561,0.201194093997435,0.198431485327111
0.394151347077563,0.186161000115562,0.570972172608539,0.166269205816994
0.724417731360170,0.139570677926154,0.848206583410427,0.107159220467172
0.937273392400706,0.070366047488108,0.987992518020485,0.030753241996117
0.095012509837637,0.189450610455068,0.281603550779259,0.182603415044924
0.458016777657227,0.169156519395002,0.617876244402644,0.149695988816577
0.755404408355003,0.124628971255534,0.865631202387832,0.095158511682493
0.944575023073232,0.062253523938648,0.989400934991650,0.027152459411754
0.076526521133497,0.152753387130726,0.227785851141645,0.149172986472604
0.373706088715420,0.142096109318382,0.510867001950827,0.131688638449177
0.636053680726515,0.118194531961518,0.746331906460151,0.101930119817240
0.839116971822219,0.083276741576705,0.912234428251326,0.062672048334109
0.963971927277914,0.040601429800387,0.993128599185095,0.017614007139152
0.064056892862606,0.127938195346752,0.191118867473616,0.125837456346828
0.315042679696163,0.121670472927803,0.433793507626045,0.115505668053726
0.545421471388840,0.107444270115966,0.648093651936976,0.097618652104114
0.740124191578554,0.086190161531953,0.820001985973903,0.073346481411080

```

0.886415527004401,0.059298584915437,0.938274552002733,0.044277438817420
 0.974728555971309,0.028531388628934,0.995187219997021,0.012341229799987

A.3.3.3. Subroutine **RULES** for integration by (i) Trapezoidal,
 (ii) Simpson, (iii) Gauss quadrature and (iv) Romberg
 methods.

Program name : **RULES.FOR**

```

C _____
C
C THIS SUBROUTINE EMPLOYS (I) TRAPEZOIDAL, (II) SIMPSON'S, (III) GAUSS-
C LEGENDRE QUADRATURE AND (IV) ROMBERG METHODS FOR INTEGRATION.
C _____
C
  SUBROUTINE RULES(F)
    COMMON /BLOK/ Z(24,24),W(24,24),T(15,15),A,B,XI,H,ER,K,KK,LL
C
    FI = 0.0
    FII = 0.0
    IF(KK .EQ. 1) THEN
C
C TRAPEZOIDAL RULE WITH AND WITHOUT END CORRECTION IS PROVOKED ...
C
      DO 10 I=1,K-1
        FI = FI + F(A + I*H)
10    CONTINUE
      FA = F(A)
      FB = F(B)
      IF(LL .EQ. 1) THEN
        XI = H*(FA + FB + 2.0*FI)/2.0
      ELSE
        FDA = (F(A+H) - FA)/H
        FDB = (F(B+H) - FB)/H
        CORR TN = H**2*(FDB - FDA)/12.0
        XI = H*(FA + FB + 2.0*FI)/2.0 - CORR TN
      ENDIF
      ELSEIF(KK .EQ. 2) THEN
C
C SIMPSON'S RULE WITH AND WITHOUT END CORRECTION IS USED ...
C
      DO 20 I=1,K-1,2
        FI = FI + F(A + I*H)
20    CONTINUE
      DO 30 I=2,K-2,2
        FII = FII + F(A + I*H)
30    CONTINUE
      FA = F(A)
      FB = F(B)
      IF(LL .EQ. 1) THEN
        XI = H*(FA + FB + 4.0*FI + 2.0*FII)/3.0
      ELSE
        FDA = (F(A+H) - FA)/H

```

```

      FDB = (F(B+H) - FB)/H
      XI = H*(14.0*(FA/2.0+FB/2.0+FII)+16.0*FI+H*(FDA-FDB))/15.0
    ENDIF
  ELSEIF(KK .EQ. 3) THEN
C
C  GAUSS QUADRATURE METHOD STARTS ...
C
    IF(MOD(K,2) .EQ. 0) THEN
      K1 = K/2
    ELSE
      K1 = K/2 + 1
    ENDIF
    DO 40 I=1,K
      IF(I .GT. K1) THEN
        IF(MOD(K,2) .EQ. 0) THEN
          Z(K,I) = -Z(K,I-K1)
          W(K,I) = W(K,I-K1)
        ELSE
          Z(K,I) = -Z(K,I-K1+1)
          W(K,I) = W(K,I-K1+1)
        ENDIF
      ENDIF
      X = (B + A)/2.0 + (B - A)*Z(K,I)/2.0
      FI = FI + W(K,I)*F(X)
40    CONTINUE
      XI = (B - A)*FI/2.0
    ELSEIF(KK .EQ. 4) THEN
C
C  ROMBERG METHOD BEGINS ...
C
      T(1,1) = (B - A)*(F(A) + F(B))/2.0
      DO 70 I=1,15
        K = 2**I
        C = (B - A)/K
        FI = 0.0
        DO 50 L=1,K-1,2
          D = A + C*L
          FI = FI + F(D)
50      CONTINUE
          T(I+1,1) = T(I,1)/2.0 + C*FI
          IF(I .EQ. 1) THEN
            T(1,2) = (4.0*T(I+1,1) - T(I,1))/3.0
          ELSE
            J = 2
60          L = 1 + 2 - J
            T(L,J) = (4**J-1)*T(L+1,J-1) - T(L,J-1))/(4**J-1 - 1.0)
            IF(J .EQ. I+1) THEN
              ERROR = (T(1,J) - T(1,J-1))/T(1,J)
              IF(ABS(ERROR) .LE. ER) THEN
                XI = T(1,J)
                RETURN
              ELSE
                GOTO 70
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF

```

```

        ENDIF
      ELSE
        J = J + 1
        GOTO 60
      ENDIF
    ENDIF
70 CONTINUE
  ENDIF
  RETURN
END
C

```

A.3.3.4. Main program that calls subroutines RULES and WEIGHT for doing numerical integration.

Program name : **MAIN.FOR**

```

C
C THIS PROGRAM COUPLED WITH THE SUBROUTINE 'RULES' CAN INTEGRATE
C ANY ANALYTIC EQUATION EMPLOYING (I) TRAPEZOIDAL (II) SIMPSON'S
C (III) GAUSS-LEGENDRE QUADRATURE AND (IV) ROMBERG METHODS.
C
C      Program Developed by: Prof. Prabir K. Chandra
C
C
C COMMON /BLOK/ Z(24,24),W(24,24),T(15,15),A,B,XI,H,ER,K,KK,LL
C DIMENSION NPANEL(14)
C CHARACTER S *16, S1 *1, SS *22
C EXTERNAL F
C DATA NPANEL /2,3,4,5,6,7,8,9,10,12,15,16,20,24/
C
C OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C KK : INDEX NUMBER [1] TRAPEZOIDAL; [2] SIMPSON'S; [3] GAUSS-
C      LEGENDRE QUADRATURE and [4] ROMBERG METHOD
C A, B, H : LOWER AND UPPER LIMITS OF THE INTEGRAL, STEPSIZE
C XI, K : THE VALUE OF THE INTEGRAL AND NUMBER OF PANELS
C
C DO 40 II=1,100
C   WRITE(*,50)
C   READ(*,*) KK
C   IF(KK .EQ. 1) THEN
C     S = 'TRAPEZOIDAL RULE'
C   ELSEIF(KK .EQ. 2) THEN
C     S = 'SIMPSON METHOD'
C   ELSEIF(KK .EQ. 3) THEN
C     S = 'GAUSS QUADRATURE'
C   ELSEIF(KK .EQ. 4) THEN
C     S = 'ROMBERG METHOD'
C   ELSE
C     WRITE(20,170)

```

```

        CLOSE(20,STATUS='KEEP')
        STOP
    ENDIF
    SS = ' '
    IF(KK .LE. 2) THEN
        WRITE(*,60) S
        READ(*,*) LL
        IF(LL .EQ. 1) THEN
            SS = 'WITHOUT END CORRECTION'
        ELSEIF(LL .EQ. 2) THEN
            SS = ' WITH END CORRECTION '
        ELSE
            GOTO 40
        ENDIF
    ENDIF
    WRITE(20,70) S,SS
    WRITE(20,80)
    IF(KK .EQ. 3) CALL WEIGHT
10  IF(KK .EQ. 4) THEN
        K = 2
        WRITE(*,90) S
        READ(*,*) A,B,ER
    ELSE
        WRITE(*,100) S
        READ(*,*) K,A,B
    ENDIF
    IF((A .GE. B) .OR. (K .LT. 1)) THEN
        WRITE(*,110)
        PAUSE
        GOTO 10
    ENDIF
C
C IF NUMBER OF PANELS SELECTED IN GAUSS QUADRATURE IS NOT PRESENT
IN
C THE PROGRAM GIVES A SUITABLE MESSAGE AND GOES BACK TO READ DATA
C
    IF(KK .EQ. 3) THEN
        DO 20 I=1,14
            IF(K .EQ. NPANEL(I)) GOTO 30
20  CONTINUE
        WRITE(*,120)
        PAUSE
        GOTO 10
    ENDIF
    IF((KK .EQ. 2) .AND. (MOD(K,2) .NE. 0)) THEN
        WRITE(*,130)
        PAUSE
        GOTO 10
    ENDIF
C
C INTEGRATION IS PERFORMED BY CALLING SUBROUTINE 'RULES'
C
    H = (B - A)/K

```

```

30  CALL RULES(F)
    IF(K .GE. 32768) THEN
        WRITE(*,140)
        PAUSE
        GOTO 10
    ENDIF
    WRITE(*,150) S,SS
    WRITE(*,80)
    WRITE(*,160) K,A,B,XI
    WRITE(20,160) K,A,B,XI
    PAUSE
40  CONTINUE
C
C ***** FORMAT STATEMENTS *****
C
50  FORMAT(15(/)1X,79('#')/1X,'#',77X,'#/1X,'#',16X,'NUMERICAL ',
1      'INTEGRATION OF CONTINUOUS FUNCTIONS',16X,'#/1X,'#',77X,
1      '#/1X,'#',30X,'MAIN OPTION MENU',31X,'#/1X,'#',77X,'#'
1      '/1X,'#',25X,'CHOOSE ANY METHOD BY NUMBER',25X,'#/1X,'#',
1      '25X,27('-'),25X,'#/1X,'#',77X,'#/1X,'#',30X,'1: TRAPE',
1      'ZOIDAL',33X,'#/1X,'#',30X,'2: SIMPSON',37X,'#/1X,'#',
1      '30X,'3: GAUSS QUADRATURE',28X,'#/1X,'#',30X,'4: ROMBERG',
1      '37X,'#/1X,'#',77X,'#/1X,'#',23X,['Type any OTHER ',
1      'number to QUIT'],23X,'#/1X,'#',77X,'#/1X,79('#')//)
60  FORMAT(20(/)32X,A16/32X,16('-')/1X,79('#')/1X,'#',77X,'#/1X,
1      '#',31X,'SECONDARY MENU',32X,'#/1X,'#',77X,'#/1X,'#',
1      '25X,'CHOOSE ANY METHOD BY NUMBER',25X,'#/1X,'#',25X,
1      '27('-'),25X,'#/1X,'#',77X,'#/1X,'#',27X,'1: WITHOUT ',
1      'END CORRECTION',25X,'#/1X,'#',27X,'2: WITH END CORREC',
1      'TION',28X,'#/1X,'#',77X,'#/1X,'#',15X,['Type any ',
1      'OTHER number to go back to MAIN MENU'],15X,'#/1X,'#',
1      '77X,'#/1X,79('#')//)
70  FORMAT(/28X,A16/28X,16('-')/25X,A22/)
80  FORMAT(8X,'No. Panel',4X,'Lower Limit',6X,'Upper Limit',7X,
1      'Integral'/8X,59('-'))
90  FORMAT(20(/)32X,A16//5X,'Give LOWER and UPPER limits and ERROR ',
1      'bound (REAL numbers) and <ENTER>.'/4X,72('-')//)
100 FORMAT(20(/)32X,A16//21X,'Give number of panels (INTEGER value)'/
1      '5X,'and LOWER and UPPER limits (REAL numbers : LOWER <',
1      'UPPER) and <ENTER>.'/15X,'Number of PANELS should be ',
1      'EVEN for SIMPSON METHOD'/10X,'Number of PANELS should ',
1      'NOT EXCEED 24 for GAUSS QUADRATURE.'/5X,70('_')//)
110 FORMAT(20(/)13X,'LOWER LIMIT can NOT be LARGER than HIGHER ',
1      'LIMIT and/or/6X,'Number of PANELS SHOULD NOT be LESS ',
1      'THAN ONE ... ENTER correct DATA.'//)
120 FORMAT(20(/)6X,'GAUSS QUADRATURE can have 2 to 10, 12,15,16,20 ',
1      'or 24 panels ONLY'/6X,64('-')//)
130 FORMAT(20(/)2X,'Number of PANELS SHOULD NOT be ODD for SIMPSON ',
1      'method ... ENTER correct DATA.'//)
140 FORMAT(20(/)3X,'Probably ROMBERG method is NOT converging ... ',
1      'Give a LARGER ERROR bound.'//)
150 FORMAT(20(/)28X,A16/28X,16('-')/25X,A22/)
160 FORMAT(9X,I5,4X,IPE15.8,2X,IPE15.8,2X,IPE15.8)

```

```
170 FORMAT(8X,59('-'))
      STOP
      END

C
C
C THE SUBROUTINE "RULES" IS INCLUDED ...
C
C
$INCLUDE:'RULES.FOR'
C
C
C THE SUBROUTINE "WEIGHT" IS INCLUDED
C
C
$INCLUDE:'WEIGHT.FOR'
C
C
C FUNCTION DEFINES THE EQUATION TO BE INTEGRATED ...
C IT IS INCLUDED IN AN EXTERNAL FILE CALLED "FCTN.FOR".
C
C
$INCLUDE:'FCTN.FOR'
C
```

Appendix A.3.3.5. A sample program for input function.
Program name : **FCTN.FOR**

```
C
C
C THE FUNCTION F(X) IS TO BE SUPPLIED BY THE USER ...
C Program name : FCTN.FOR
C
C
FUNCTION F(X)
C
F = 1.0/X
RETURN
END
C
```

Appendix A.3.3.6. Zeros and weights of Legendre polynomials.

m	$\pm z_k$	w_k	$\pm z_k$	w_k
2	0.577350269189626	1.000000000000000		
3	0.000000000000000	0.888888888888889	0.774596669241483	0.555555555555556

4	0.339981043584856	0.652145154862546	0.861136311594053	0.347854845137454
5	0.000000000000000	0.568888888888889	0.538469310105683	0.478628670499366 0.906179845938664
6	0.238619186083197	0.467913934572691	0.661209386466265	0.360761573048139 0.932469514203152
		0.171324492379170		

contd....

Appendix A.3.3.6. Zeros and weights of Legendre polynomials - contd.

m	$\pm z_k$	w_k	$\pm z_k$	w_k
7	0.000000000000000 0.741531185599394	0.417959183673469 0.279705391489277	0.405845151377397 0.949107912342759	0.381830050505119 0.129484966168870
8	0.183434642495650 0.796666477413627	0.362683783378362 0.222381034453374	0.525532409916329 0.960289856497536	0.313706645877887 0.101228536290376
9	0.000000000000000 0.613371432700590 0.968160239507626	0.330239355001260 0.260610696402935 0.081274388361574	0.324253423403809 0.836031107326636	0.312347077040003 0.180648160694857
10	0.148874338981631 0.679409568299024 0.973906528517172	0.295524224714753 0.219086362515982 0.066671344308688	0.433395394129247 0.865063366688985	0.269266719309996 0.149451349150581
12	0.125233408511469 0.587317954286617 0.904117256370475	0.249147045813403 0.203167426723066 0.106939325995318	0.367831498998180 0.769902674194305 0.981560634246719	0.233492536538355 0.160078328543346 0.047175336386512
15	0.000000000000000 0.394151347077563 0.724417731360170 0.937273392400706	0.202578241925561 0.186161000115562 0.139570677926154 0.070366047488108	0.201194093997435 0.570972172608539 0.848206583410427 0.987992518020485	0.198431485327111 0.166269205816994 0.107159220467172 0.030753241996117
16	0.095012509837637 0.458016777657227 0.755404408355003 0.944575023073232	0.189450610455068 0.169156519395002 0.124628971255534 0.062253523938648	0.281603550779259 0.617876244402644 0.865631202387832 0.989400934991650	0.182603415044924 0.149695988816577 0.095158511682493 0.027152459411754
20	0.076526521133497 0.373706088715420 0.636053680726515 0.839116971822219 0.963971927277914	0.152753387130726 0.142096109318382 0.118194531961518 0.083276741576705 0.040601429800387	0.227785851141645 0.510867001950827 0.746331906460151 0.912234428251326 0.993128599185095	0.149172986472604 0.131688638449177 0.101930119817240 0.062672048334109 0.017614007139152
24	0.064056892862606 0.315042679696163	0.127938195346752 0.121670472927803	0.191118867473616 0.433793507626045	0.125837456346828 0.115505668053726

0.545421471388840 0.107444270115966 0.648093651936976 0.097618652104114
0.740124191578554 0.086190161531953 0.820001985973903 0.073346481411080
0.886415527004401 0.059298584915437 0.938274552002733 0.044277438817420
0.974728555971309 0.028531388628934 0.995187219997021 0.012341229799987

APPENDIX A.3.4. Program listings for solution of linear algebraic equations.

A.3.4.1. Subroutine SOLVE for solution of linear algebraic equations by the methods of (1) Gauss, (2) Gauss-Jordan, (3) Matrix inversion and (4) Gauss-Seidel iteration.

Program name : **SOLVE.FOR**

```

C _____
C
C THIS SUBROUTINE EMPLOYS (I) GAUSS, (II) GAUSS-JORDAN, (III) MATRIX
C INVERSION AND (IV) GAUSS-SEIDEL ITERATIVE METHODS FOR SOLUTION.
C _____
C
  SUBROUTINE SOLVE
    COMMON /BLOK/ A(50,51),AI(50,51),X(51),R(51),ERROR,XX,N,KK,K
    DOUBLE PRECISION A,AI,X,R,ERROR,TEMP,AA,SUM,XK,XX

    IF(KK .EQ. 1) THEN
C
C GAUSS ELIMINATION METHOD STARTS ...
C LAST COLUMN ELEMENTS OF AUGMENTED MATRIX ARE FORMED ...
C
      DO 10 I=1,N
        A(I,N+1) = R(I)
10      CONTINUE
C
      DO 60 K=1,N-1
        JJ = K + 1
        II = K
        DO 20 I=JJ,N
          IF(DABS(A(I,K)) .GT. DABS(A(II,K)))II=I
20        CONTINUE
          IF(II .NE. K) THEN
            DO 30 J1=K,N+1
              TEMP = A(K,J1)
              A(K,J1) = A(II,J1)
              A(II,J1) = TEMP
30            CONTINUE
          ENDIF
          DO 50 I=JJ,N
            AA = A(I,K)/A(K,K)
            DO 40 J=JJ,N+1
              A(I,J) = A(I,J) - A(K,J)*AA
40            CONTINUE
50          CONTINUE
60        CONTINUE
C
C BACK SUBSTITUTION TO CALCULATE X(I)
C

```

```

      DO 80 K=N,1,-1
        SUM = 0.0
        DO 70 I=1,N-K
          SUM = SUM + A(K,I+K)*X(I+K)
70      CONTINUE
        X(K) = (A(K,N+1) - SUM)/A(K,K)
80      CONTINUE
      ELSEIF(KK .EQ. 2) THEN
C
C  GAUSS-JORDAN ELIMINATION METHOD BEGINS ...
C
      DO 120 K=1,N
        AA = A(K,K)
        IF(AA .EQ. 0.0) AA=1.0D-10
        R(K) = R(K)/AA
        DO 90 L=1,N
          A(K,L) = A(K,L)/AA
90      CONTINUE
        A(K,K) = 0.0
        DO 110 I=1,N
          AA = A(I,K)
          R(I) = R(I) - R(K)*AA
          X(I) = R(I)
          DO 100 J=1,N
            A(I,J) = A(I,J) - A(K,J)*AA
100     CONTINUE
110     CONTINUE
120     CONTINUE
      ELSEIF(KK .EQ. 3) THEN
C
C  MATRIX INVERSION METHOD STARTS ...
C  IDENTITY MATRIX IS FORMED.
C
      DO 140 I=1,N
        DO 130 J=1,N
          AI(I,J) = 0.0
          IF(I .EQ. J) AI(I,J)=1.0
130     CONTINUE
140     CONTINUE
      DO 180 K=1,N
        AA = A(K,K)
        IF(AA .EQ. 0.0) AA=1.0D-10
        DO 150 L=1,N
          A(K,L) = A(K,L)/AA
          AI(K,L) = AI(K,L)/AA
150     CONTINUE
        A(K,K) = 0.0
        DO 170 I=1,N
          AA = A(I,K)
          DO 160 J=1,N
            A(I,J) = A(I,J) - A(K,J)*AA
            AI(I,J) = AI(I,J) - AI(K,J)*AA
160     CONTINUE

```

```

170     CONTINUE
180     CONTINUE
C
C   VALUES OF X(I) ARE CALCULATED ...
C
      DO 200 I=1,N
        X(I) = 0.0
        DO 190 J=1,N
          X(I) = X(I) + A(I,J)*R(J)
190     CONTINUE
200     CONTINUE
      ELSEIF(KK .EQ. 4) THEN
C
C   GAUSS-SEIDEL ITERATIVE METHOD BEGINS ...
C   FIRST GUESSES ON X ARE MADE ...
C
      DO 210 I=1,N
        X(I) = XX
210     CONTINUE
C
      DO 240 K=1,1000
        SUM = 0.0
        DO 230 I=1,N
          XK = X(I)
          IF(A(I,I) .EQ. 0.0) A(I,I)=1.0D-10
          X(I) = R(I)/A(I,I)
          DO 220 J=1,N
            IF(I .NE. J) THEN
              X(I) = X(I) - A(I,J)*X(J)/A(I,I)
            ENDIF
          IF(DABS(X(I)) .GE. 1.0D32) THEN
            XX = -1001.0
            RETURN
          ENDIF
220     CONTINUE
          AA = (X(I) - XK)/X(I)
          SUM = SUM + DABS(AA)
230     CONTINUE
        IF(SUM .LE. ERROR) RETURN
240     CONTINUE
      ENDIF
      RETURN
      END
C

```

A.3.4.2. Main program for solving linear algebraic equations.

Program name : **MAIN.FOR**

```

C
C   THIS PROGRAM COUPLED WITH THE SUBROUTINE 'SOLVE' CAN SOLVE A
C   NUMBER OF LINEAR ALGEBRAIC EQUATIONS EMPLOYING (I) GAUSS (II)

```

C GAUSS-JORDAN (III) MATRIX INVERSION AND (IV) GAUSS-SEIDEL
C ITERATION METHODS.

C
C Program Developed by: Prof. Prabir K. Chandra

C
C COMMON /BLOK/ A(50,51),AI(50,51),X(51),R(51),ERROR,XX,N,KK,K
C CHARACTER S *25
C DOUBLE PRECISION A,AI,X,R,ERROR,XX
C
C OPEN(10,FILE='MATRIX.IN',STATUS='OLD')
C OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C KK : INDEX NUMBER [1] GAUSS ELIMINATION; [2] GAUSS-JORDAN
C ELIMINATION; [3] MATRIX INVERSION and [4] GAUSS-
C SEIDEL ITERATIVE METHODS
C A, X, R : COEFFICIENT MATRIX, SOLUTION AND CONSTANT VECTORS
C AI : INVERSE OF MATRIX [A]
C N : SIZE OF THE SQUARE MATRIX or NUMBER OF EQUATIONS.
C
C DO 20 II=1,100
C WRITE(*,30)
C READ(*,*) KK
C IF(KK .EQ. 1) THEN
C S = 'GAUSS ELIMINATION METHOD'
C ELSEIF(KK .EQ. 2) THEN
C S = 'GAUSS-JORDAN METHOD'
C ELSEIF(KK .EQ. 3) THEN
C S = 'MATRIX INVERSION METHOD'
C ELSEIF(KK .EQ. 4) THEN
C S = 'GAUSS-SEIDEL ITERATION'
C ELSE
C WRITE(20,110)
C CLOSE(20,STATUS='KEEP')
C STOP
C ENDIF
C WRITE(20,50) S
C READ(10,*) N,((A(I,J),J=1,N),I=1,N),(R(I),I=1,N)
C IF(KK .LE. 2) WRITE(20,70) N
10 IF(KK .EQ. 4) THEN
C WRITE(*,60) S
C READ(*,*) XX,ERROR
C ENDIF
C
C SUBROUTINE 'SOLVE' IS CALLED ...
C
C CALL SOLVE
C IF(KK .EQ. 4) THEN
C IF((XX .EQ. -1001.0) .OR. (K .EQ. 1001)) THEN
C WRITE(*,40)
C PAUSE
C GOTO 10
C ENDIF

```

WRITE(*,90) S
WRITE(*,70) N
WRITE(*,80) K
WRITE(20,80) K
ELSEIF(KK .EQ. 3) THEN
WRITE(*,90) S
WRITE(*,85)
WRITE(20,85)
WRITE(*,100) ((AI(I,J),J=1,N),I=1,N)
WRITE(20,100) ((AI(I,J),J=1,N),I=1,N)
PAUSE
WRITE(*,70) N
WRITE(20,70) N
ELSE
WRITE(*,90) S
WRITE(*,70) N
ENDIF
WRITE(*,100) (X(I),I=1,N)
WRITE(20,100) (X(I),I=1,N)
PAUSE
REWIND 10
20 CONTINUE
C
C ***** FORMAT STATEMENTS *****
C
30 FORMAT(20(/)1X,79('#')1X,'#',77X,'#/1X,'#',19X,'SOLUTION OF ',
1 'LINEAR ALGEBRAIC EQUATIONS',20X,'#/1X,'#',77X,
1 '#/1X,'#',30X,'MAIN OPTION MENU',31X,'#/1X,'#',77X,'#'
1 '/1X,'#',24X,'CHOOSE ANY METHOD BY NUMBER',26X,'#/1X,'#',
1 24X,27('-'),26X,'#/1X,'#',77X,'#/1X,'#',25X,'1: GAUSS',
1 'ELIMINATION',32X,'#/1X,'#',25X,'2: GAUSS-JORDAN ELI',
1 'MINATION',25X,'#/1X,'#',25X,'3: MATRIX INVERSION',33X,
1 '#/1X,'#',25X,'4: GAUSS-SEIDEL ITERATION',27X,'#/1X,'#',
1 77X,'#/1X,'#',23X,['Type any OTHER number to QUIT'],23X,
1 '#/1X,'#',77X,'#/1X,79('#')///)
40 FORMAT(20(/)2X,'Scheme NOT converging. TRY with new initial ',
1 'GUESS and/or larger ERROR bound.'////)
50 FORMAT(/28X,A25/28X,25('-'))
60 FORMAT(20(/)28X,A25/28X,25('-')//4X,'Give the first GUESS of X',
1 ' and an ERROR bound (REAL numbers) and <ENTER>.'////)
70 FORMAT(/19X,'Solution of ',I3,' linear algebraic equations'/19X,
1 42('-'))
80 FORMAT(18X,['Number of iterations for CONVERGENCE = ',I4,']/')
85 FORMAT(24X,'ROW-WISE INVERTED MATRIX ELEMENTS'/24X,33('-'))
90 FORMAT(20(/)28X,A25/28X,25('-'))
100 FORMAT(6X,1PD15.8,2X,1PD15.8,2X,1PD15.8,2X,1PD15.8)
110 FORMAT(6X,68('-'))
STOP
END
C
C
C THE SUBROUTINE "SOLVE.FOR" IS INCLUDED (LISTED IN APPENDIX A.3.4.1)
C

```

```
C
$INCLUDE:'SOLVE.FOR'
C
```

A.3.4.3. A sample data input file for the program "MAIN.FOR".
 File name : **MATRIX.IN**

```
8
17.0,1.0,4.0,3.0,-1.0,2.0,3.0,-7.0,2.0,10.0,-1.0,7.0,-2.0,1.0,1.0,-4.0,
-1.0,1.0,-8.0,2.0,-5.0,2.0,-1.0,1.0,2.0,4.0,1.0,-11.0,1.0,3.0,4.0,-1.0,
1.0,3.0,1.0,7.0,-15.0,1.0,-2.0,4.0,-2.0,1.0,7.0,-1.0,2.0,12.0,-1.0,8.0,
3.0,4.0,5.0,1.0,2.0,8.0,-19.0,2.0,5.0,1.0,1.0,1.0,-1.0,1.0,-7.0,10.0
71.0,43.0,-11.0,-37.0,-61.0,52.0,-73.0,21.0
```

A.3.4.4. Subroutine TRIDGL for solving tridiagonal matrix.
 Program name : **TRIDGL.FOR**

```
C
C
C SUBROUTINE "TRIDGL" TO SOLVE A SET OF LINEAR EQUATIONS IN TRIDIAGONAL
C MATRIX FORM.
C
C
C SUBROUTINE TRIDGL
COMMON /BLOK/ A(101),B(101),D(101),R(101),X(101),N
DOUBLE PRECISION A,B,D,R,X,AA
C
C A(N) = 0.0
C B(1) = 0.0
C B(N) = B(N)/D(N)
C R(N) = R(N)/D(N)
C
C DO 10 I=2,N
C   J = N - I + 2
C   AA = 1.0/(D(J-1) - B(J)*A(J-1))
C   B(J-1) = B(J-1)*AA
C   R(J-1) = (R(J-1) - A(J-1)*R(J))*AA
10 CONTINUE
C
C BACK SUBSTITUTION STARTS ...
C
C X(1) = R(1)
C DO 20 I=2,N
C   X(I) = R(I) - B(I)*X(I-1)
20 CONTINUE
C
C RETURN
C END
C
```

A.3.4.5. Main program to solve tridiagonal matrix by calling subroutine TRIDGL.
 Program name : BANDED.FOR

```

C
C
C   THIS PROGRAM SOLVES A BANDED MATRIX CALLING THE SUBROUTINE "TRIDGL"
C
C
C   COMMON /BLOK/ A(101),B(101),D(101),R(101),X(101),N
C   DOUBLE PRECISION A,B,D,R,X
C
C   OPEN(10,FILE='MATRIX.TRI',STATUS='OLD')
C   OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C   WRITE(*,30)
C   WRITE(20,30)
C
C   D, A, B and N: DIAGONAL, UPPER, LOWER ELEMENTS and ROW OR
C   COLUMN SIZE OF TRIANGULAR MATRIX.
C   R, X: ELEMENTS OF CONSTANT VECTOR AND SOLUTION VECTOR
C
C   DO 10 K=1,100
C     READ(10,*,END=20) N,(A(I),I=1,N),(B(I),I=1,N),
1     (D(I),I=1,N),(R(I),I=1,N)
C
C     CALL TRIDGL
C
C     WRITE(*,40) K,(X(I),I=1,N)
C     WRITE(20,40) K,(X(I),I=1,N)
C     PAUSE
10  CONTINUE
20  WRITE(20,50)
C
C     ***** FORMAT STATEMENT *****
C
30  FORMAT(20(/)20X,40('#')//25X,'SOLUTION OF TRIDIAGONAL ',
1    'MATRIX'//20X,40('#'))
40  FORMAT(/30X,'DATA SET NUMBER = ',I2/30X,20('-')/(6X,1PD15.8,
1    2X,1PD15.8,2X,1PD15.8,2X,1PD15.8))
50  FORMAT(6X,66('_'))
C
C   STOP
C   END
C
C
C   THE SUBROUTINE "TRIDGL.FOR" IS INCLUDED (LISTED IN APPENDIX A.3.4.3)
C

```

C

\$INCLUDE:"TRIDGL.FOR"

C

A.3.4.6. A sample data input file for the program "BANDED.FOR".

File name : **MATRIX.TRI**

10

1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0

1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0

-4.0,-4.0,-4.0,-4.0,-4.0,-4.0,-4.0,-4.0,-4.0,-4.0

-27.0,-15.0,-15.0,-15.0,-15.0,-15.0,-15.0,-15.0,-15.0,-15.0

APPENDIX A.3.5. List of PROGRAMS used in Regression Analysis.

A.3.5.1. Subroutine REGMLT for multilinear regression.

Program name : **REGMLT.FOR**

```
C _____
C
C SUBROUTINE "REGMLT" TO EXECUTE MULTILINEAR REGRESSION USING
C VALUES OF Y, X1, X2, X3 etc, CONSTRUCTING AN EQUATION
C  $Y = B_0 + B_1.X_1 + B_2.X_2 + B_3.X_3 + \dots$  ALSO WILL BE CALCULATED
C VALUES OF COEFICIENTS B0, B1, B2, B3 etc. AND THE CORRELATION
C COEFFICIENT, R.
C _____
C
  SUBROUTINE REGMLT(X,GAUSS)
    DIMENSION X(12,100)
    COMMON /CONS/ N,K,MM,L,KT
    COMMON /BLK1/ A(12,12),Y(100),YHAT(100),B(12),G(12),YSSQ,SDERR,
1      PCERR,R
C
    K1 = K + 1
    DO 30 I=1,K1
      B(I) = 0.0
      DO 20 J=1,K1
        A(I,J) = 0.0
20    CONTINUE
30    CONTINUE
      DO 60 I=1,K1
        DO 50 J=1,K1
          DO 40 L1=1,N
            YHAT(L1) = 0.0
            IF((I .EQ. 1) .AND. (J .EQ. 1)) THEN
              A(I,J) = FLOAT(N)
            ELSE IF((I .GE. 2) .AND. (J .EQ. 1)) THEN
              A(I,J) = A(I,J) + X(J-1,L1)*X(J-1,L1)
            ELSE IF((I .EQ. 1) .AND. (J .GT. 1)) THEN
              A(I,J) = A(I,J) + X(J-1,L1)
            ELSE IF((I .GE. 2) .AND. (J .GT. 1)) THEN
              A(I,J) = A(I,J) + X(I-1,L1)*X(J-1,L1)
            ENDIF
40          CONTINUE
            IF((I .GE. 1) .AND. (J .NE. 1)) THEN
              A(J,I) = A(I,J)
            ENDIF
50          CONTINUE
60          CONTINUE
C
      DO 80 I=1,K1
        DO 70 J=1,N
          IF(I .EQ. 1) THEN
            B(I) = B(I) + Y(J)
```

```

        G(I) = B(I)
    ELSE
        B(I) = B(I) + X(I-1,J)*Y(J)
        G(I) = B(I)
    ENDIF
70    CONTINUE
80    CONTINUE
C
    CALL GAUSS(K1)
C
    YAV = 0.0
    S1 = 0.0
    YSSQ = 0.0
C
    DO 90 I=1,K1
        S1 = S1 + B(I)*G(I)
90    CONTINUE
C
    DO 110 I=1,N
        DO 100 J=1,K
            YHAT(I) = YHAT(I) + B(J+1)*X(J,I)
            YH = YHAT(I)
100    CONTINUE
            YHAT(I) = B(1) + YH
            YAV = YAV + Y(I)/FLOAT(N)
            YSSQ = YSSQ + (YHAT(I) - Y(I))*(YHAT(I) - Y(I))
110    CONTINUE
        SDERR = SQRT(YSSQ/FLOAT(N))
        PCERR = 100.0*SDERR/YAV
C
        S2 = 0.0
        S3 = 0.0
        DO 120 I=1,N
            S2 = S2 + (Y(I) - YAV)*(Y(I) - YAV)
            S3 = S3 + Y(I)
120    CONTINUE
C
        R = SQRT(ABS(S1 - S3*S3/FLOAT(N))/S2)
C
        RETURN
    END
C

```

A.3.5.2. Subroutine REGNON for asymptotic and logistic regressions.
Program name : **REGNON.FOR**

```

C_____
C
C SUBROUTINE "REGNON" TO EXECUTE THE NONLINEAR REGRESSION USING
C N-VALUES OF X AND Y IN THE FOLLOWING FORMS:
C

```

```

C   When M=1 : Y = B0 + B1*p^X or Y = B0 + B1.e^B2X
C   When M=2 : Y = B0/(1 + B1*p^X) or Y= B0/(1 + B1.e^B2X)
C   where, B2 = ln(p)
C
C
SUBROUTINE REGNON(MX,M,GAUSS,REGMLT,R1)
COMMON /CONS/ N,K,MM,L,KT
COMMON /BLK1/ A(12,12),Y(100),YHAT(100),B(12),G(12),YSSQ,SDERR,
1          PCERR,R
COMMON /BLK2/ X(12,100),X1(12,100)
C
IF(MX .EQ. 1) THEN
  RA = 0.0
  RB = 0.0
  DO 10 I=2,N
    RA = RA - ((-1)**I)*Y(I)
    RB = RB + ((-1)**(I-1))*Y(I-1)
10  CONTINUE
  IF(MOD(N,2) .EQ. 0) THEN
    R1 = (RA + Y(N-1))/(RB + Y(N-2))
  ELSE
    R1 = RA/RB
  ENDIF
  DO 20 I=1,N
    IF(Y(I) .EQ. 0.0) Y(I)=1.0E-5
    IF(M .EQ. 2) THEN
      Y(I) = 1.0/Y(I)
    ELSE
      Y(I) = Y(I)
    ENDIF
20  CONTINUE
  IF((R1 .GE. 1.0) .OR. (R1 .LE. 0.0)) R1=0.99
  RETURN
ENDIF
C
YSSQ1 = 1.0E10
DO 60 K3=1,150
  MM = K3
  DO 40 I=1,K
    DO 30 J=1,N
      IF(I .LE. 1) THEN
        X1(I,J) = R1**(X(I,J))
      ELSE
        X1(I,J) = X(1,J)*R1**(X(1,J)-1.0)
      ENDIF
30    CONTINUE
40  CONTINUE
C
CALL REGMLT(X1,GAUSS)
C
IF(R1 .LE. 0.0) RETURN
ERRO = 1.0E-06
ERRO1 = ABS(YSSQ - YSSQ1)

```

```

IF((ERRO1 .LE. ERRO) .AND. (M .EQ. 2)) THEN
  B(2) = B(2)/B(1)
  B(1) = 1.0/B(1)
  B(3) = ALOG(R1)
  DO 50 I=1,N
    YHAT(I) = 1.0/YHAT(I)
    Y(I) = 1.0/Y(I)
50  CONTINUE
  RETURN
ELSEIF((ERRO1 .LE. ERRO) .AND. (M .NE. 2)) THEN
  B(3) = ALOG(R1)
  RETURN
ENDIF
YSSQ1 = YSSQ
R1 = R1 + B(3)/B(2)
IF(R1 .LE. 0.0) RETURN
60 CONTINUE
C
  RETURN
END
C

```

A.3.5.3. Subroutine GAUSS for solving linear, simultaneous algebraic equations.

Program name : **GAUSS.FOR**

```

C
C
C SUBROUTINE "GAUSS" USES THE METHOD OF GAUSS-JORDAN ELIMINATION
C TO SOLVE LINEAR ALGEBRAIC EQUATIONS SIMULTANEOUSLY.
C
C
C SUBROUTINE GAUSS(N)
  COMMON /BLK1/ A(12,12),Y(100),YHAT(100),B(12),G(12),YSSQ,SDERR,
  I PCERR,R
C
C A : THE ARRAY OF THE SQUARE MATRIX WHICH IS TO BE SOLVED.
C N : DIMENSION OF THE ROW OR COLUMN OF [A].
C
  DO 40 K=1,N
    AA = A(K,K)
    IF(AA .EQ. 0.0) AA=1.0E-04
    B(K) = B(K)/AA
    DO 10 L=1,N
      A(K,L) = A(K,L)/AA
10  CONTINUE
    DO 30 I=1,N
      IF(I .NE. K) THEN
        AA = A(I,K)
        B(I) = B(I) - B(K)*AA
        DO 20 J=1,N
          A(I,J) = A(I,J) - A(K,J)*AA

```

```

20      CONTINUE
      ENDIF
30      CONTINUE
40      CONTINUE
C
      RETURN
      END
C

```

A.3.5.4. Program segment for providing input data.

Program name : **INCLUDE**

```

C
C
C This program segment is common to all the main programs.
C
C
10 WRITE(*,110)
   READ(*,120) T
   IF(T.EQ. 'N') THEN
      WRITE(*,130)
      READ(*,140) FI
      OPEN(10,FILE=FI,STATUS='OLD')
      WRITE(*,150)
      READ(*,140) FO
      OPEN(20,FILE=FO,STATUS='NEW')
      GOTO 30
   ELSEIF(T.EQ. 'Y') THEN
      WRITE(*,160)
      READ(*,140) FI
      OPEN(10,FILE=FI,STATUS='NEW')
      WRITE(*,150)
      READ(*,140) FO
      OPEN(20,FILE=FO,STATUS='NEW')
      GOTO 20
   ELSE
      WRITE(*,170)
      GOTO 10
   ENDIF
C
C      ***** TYPE IN INPUT DATA *****
C
20 WRITE(*,180)
   READ(*,*) N,NVAR,(Y(I),I=1,N),((X(I,J),J=1,N),I=1,NVAR)
   WRITE(10,190) N,(Y(I),I=1,N),((X(I,J),J=1,N),I=1,NVAR)
   WRITE(*,200)
   READ(*,120) T
   IF(T.EQ. 'Y') .OR. (T.EQ. 'y') GOTO 20
   REWIND 10
30 CONTINUE

```

```

C
C ***** FORMAT STATEMENTS *****
C
110 FORMAT(/2X,'Type [Y] to READ data from SCREEN and [N] if,
      1      ' file already exists and <ENTER>.')
120 FORMAT(A1)
130 FORMAT(/2X,'Specify INPUT file that already exists (within',
      1      ' 18 characters) and <ENTER>.'/)
140 FORMAT(A18)
150 FORMAT(/2X,'Specify OUTPUT file (within 18 characters) and',
      1      ' <ENTER>.'/)
160 FORMAT(/2X,'Specify INPUT file within 18 characters to WRITE',
      1      ' DATA from SCREEN and <ENTER>.'/)
170 FORMAT(/2X,'Type CAPITAL letter (Y/N) and <ENTER>.'/)
180 FORMAT(/1X,60('-')/2X,'Give N,NVAR,Y-values,X-values and ',
      1      '<ENTER>.'/2X,'N: No. of DATA POINTS; NVAR: No. of ',
      2      'INDEPENDENT variables'/1X,60('-')/)
190 FORMAT(1X,I3/5(1X,E15.8))
200 FORMAT(/2X,'If have more DATA, press [Y] and <ENTER>.'/2X,
      1      'When all DATA exhausted, press [N] and <ENTER>.'/)
C
C

```

A.3.5.5. Main program for linear regression.

Program name: **MAIN.FOR**

```

C
C
C      Program developed by Prof. Prabir K. Chandra
C
C
C      PROGRAM MAIN
C
      COMMON /CONS/ N,K,MM,L,KT
      COMMON /BLK1/ A(12,12),Y(100),YHAT(100),B(12),G(12),YSSQ,SDERR,
      1      PCERR,R
      COMMON /BLK2/ X(12,100),X1(12,100)
      COMMON /BLK3/ LL,L1,NN,JJ,M,W
      CHARACTER *18 FI,FO
      CHARACTER *1 T,Q
      EXTERNAL GAUSS,REGMLT
C
C      ##### DATA INPUT #####
C
$INCLUDE:'INCLUDE'
C
      NN = 1
      WRITE(20,210)
      K = 1
      50 READ(10,*,END=80) N,(Y(I),I=1,N),(X(1,J),J=1,N)

```

```

CALL REGMLT(X,GAUSS)
WRITE(*,240) FI,FO,NN,(YHAT(I),Y(I),I=1,N)
WRITE(20,250)FI,FO,NN,(YHAT(I),Y(I),I=1,N)
WRITE(*,260) (B(I),I=1,K+1)
WRITE(20,260)(B(I),I=1,K+1)
WRITE(*,270) K,N,SDERR,PCERR,R
WRITE(20,270)K,N,SDERR,PCERR,R
WRITE(*,280) NN
READ(*,120) T
IF((T.EQ. 'N').OR. (T.EQ. 'n')) GOTO 80
NN = NN + 1
GOTO 50
80 WRITE(*,290)
WRITE(20,300)

C
C ***** FORMAT STATEMENTS *****
C
210 FORMAT(2X,78('_')/2X,78('-')//30X,'LINEAR REGRESSION'
1      /30X,17('-')//32X,'Y = B0 + B1.X'/)
240 FORMAT(10(/)2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,'/',
1      A18,3X,'DATA set #:',I3//24X,'Dependent variables (YHAT',
2      ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
3      E15.8,1X,E15.8))
250 FORMAT(2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,'/',
1      A18,3X,'DATA set #:',I3//24X,'Dependent variables (YHAT',
2      ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
3      E15.8,1X,E15.8))
260 FORMAT(4X,'REGRESSION coefficients'/4X,23('-')/
1      (1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8))
270 FORMAT(//2X,'No. of independent variables = ',I2,'. No. of,
1      ' DATA points = ',I3/2X,'SDERR = ',E15.8,'. Percent',
2      ' ERROR = ',F8.4,'. R = ',E15.8,'./)
280 FORMAT(//27X,'DATA set #: ',I3,' was read.'/10X,'If no more ',
1      'DATA to read Type [N], otherwise [Y] and <ENTER>.'/)
290 FORMAT(/////4X,73('#')/4X,'#',19X,'THANK you for using',
1      ' this package.',19X,'#'/4X,'#',71X,'#'/4X,'#',4X,
1      'REG2000 package has been developed by : Prof. Prabir ',
1      'K. Chandra',4X,'#'/4X,'# DCA/ESAL, 37200-000 Lavras, ',
1      'Minas Gerais, BRAZIL. FAX : (035)829-1100.#'/
1      4X,73('#')/10(/))
300 FORMAT(//2X,'SDERR: Standard ERROR = SQRT(((YHAT-Y)^2)/N).',
1      ' % ERROR = 100*SDERR/YM./2X,'YM: Mean of Y-',
2      'values, YHAT: Calculated VALUE of Y, R: Correla',
3      'tion coefficient.'/2X,78('_'))
      STOP
      END
C
$INCLUDE:'REGMLT.FOR'
CINCLUDE:'GAUSS.FOR'
C

```


A.3.5.6. Main program for multilinear regression

Program name : **MAIN.FOR**

```

C _____
C
C      Program developed by Prof. Prabir K. Chandra
C _____
C
C      PROGRAM MAIN
C
C      COMMON /CONS/ N,K,MM,L,KT
C      COMMON /BLK1/ A(12,12),Y(100),YHAT(100),B(12),G(12),YSSQ,SDERR,
1      PCERR,R
C      COMMON /BLK2/ X(12,100),X1(12,100)
C      COMMON /BLK3/ LL,L1,NN,JJ,M,W
C      CHARACTER *18 FI,FO
C      CHARACTER *1 T,Q
C      EXTERNAL GAUSS,REGMLT
C
C      ##### DATA INPUT #####
C
C      $INCLUDE:'INCLUDE'
C
C      NN = 1
40  K = 2
C      WRITE(20,210)
C      WRITE(*,220)
C      WRITE(*,*)'          (MAXIMUM independent variables = 10)'
C      WRITE(*,*)
C      READ(*,*) KT
C      READ(10,*,END=60) N,(Y(I),I=1,N),((X(I,J),J=1,N),I=1,KT)
C      KL = N - 1
C      IF(KT .GT. KL) THEN
C          WRITE(*,310) KL,NN
C          READ(*,120) T
C          IF((T .EQ. 'N') .OR. (T .EQ. 'n')) GOTO 60
C          NN = NN + 1
C          GOTO 40
C      ENDIF
50  CALL REGMLT(X,GAUSS)
C      WRITE(*,240) FI,FO,NN,(YHAT(I),Y(I),I=1,N)
C      WRITE(20,250)FI,FO,NN,(YHAT(I),Y(I),I=1,N)
C      WRITE(*,260) (B(I),I=1,K+1)
C      WRITE(20,260)(B(I),I=1,K+1)
C      WRITE(*,270) K,N,SDERR,PCERR,R
C      WRITE(20,270)K,N,SDERR,PCERR,R
C      IF(K .LT. KT) THEN
C          K = K + 1
C          PAUSE
C          GOTO 50
C      ENDIF
C      WRITE(*,280) NN

```

```

      READ(*,120) T
      IF((T.EQ. 'N') .OR. (T.EQ. 'n')) GOTO 60
      NN = NN + 1
      GOTO 40
60  WRITE(*,290)
      WRITE(20,300)

C
C  *****  FORMAT STATEMENTS  *****
C
210  FORMAT(2X,78('_')/2X,78('-')/28X,'MULTILINEAR REGRESSION'
      1      /28X,23('-')/26X,'Y = B0 + B1.X1 + B2.X2 + .../')
220  FORMAT(/11X,'Give the number of independent variables.'/)
240  FORMAT(10(/)2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,'/',
      1      A18,3X,'DATA set #:',I3/24X,'Dependent variables (YHAT',
      2      ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
      3      E15.8,1X,E15.8))
250  FORMAT(/2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,'/',
      1      A18,3X,'DATA set #:',I3/24X,'Dependent variables (YHAT',
      2      ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
      3      E15.8,1X,E15.8))
260  FORMAT(/4X,'REGRESSION coefficients'/4X,23('-')/
      1      (1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8))
270  FORMAT(/2X,'No. of independent variables = ',I2,'. No. of',
      1      ' DATA points =',I3/2X,'SDERR =',E15.8,'. Percent',
      2      ' ERROR =',F8.4,'. R =',E15.8,'/')
280  FORMAT(/27X,'DATA set #: ',I3,' was read.'/10X,'If no more ',
      1      'DATA to read Type [N], otherwise [Y] and <ENTER>.'/)
290  FORMAT(/////4X,73('#')/4X,'#',19X,'THANK you for using',
      1      ' this package.',19X,'#'/4X,'#',71X,'#'/4X,'#',4X,
      1      'REG2000 package has been developed by : Prof. Prabir ',
      1      'K. Chandra',4X,'#'/4X,'# DCA/ESAL, 37200-000 Lavras, ',
      1      'Minas Gerais, BRAZIL. FAX : (035)829-1100.#'/
      1      4X,73('#')/10(/))
300  FORMAT(/2X,'SDERR: Standard ERROR = SQRT(((YHAT-Y)^2)/N).',
      1      ' % ERROR = 100*SDERR/YM./2X,'YM: Mean of Y-',
      2      'values, YHAT: Calculated VALUE of Y, R: Correla',
      3      'tion coefficient.'/2X,78('_'))
310  FORMAT(/20X,'Insufficient DATA points. MAXIMUM K = ',I2,'.'/
      1      /27X,'DATA set #: ',I3,' was read.'/10X,'If no more ',
      2      'DATA to read Type [N], otherwise [Y] and <ENTER>.'/)
      STOP
      END

C
$INCLUDE:'REGMLT.FOR'
$INCLUDE:'GAUSS.FOR'
C

```

A.3.5.7. Main program for trigonometric regression.

Program name : **MAIN.FOR**

C_____

```

C
C      Program developed by Prof. Prabir K. Chandra
C
C
C      PROGRAM MAIN
C
C      COMMON /CONS/ N,K,MM,L,KT
C      COMMON /BLK1/ A(12,12),Y(100),YHAT(100),B(12),G(12),YSSQ,SDERR,
1      PCERR,R
C      COMMON /BLK2/ X(12,100),X1(12,100)
C      COMMON /BLK3/ LL,L1,NN,JJ,M,W
C      CHARACTER *18 FI,FO
C      CHARACTER *1 T,Q
C      EXTERNAL GAUSS,REGMLT
C
C      ##### DATA INPUT #####
C
C $INCLUDE:'INCLUDE'
C
C      NN = 1
40 K = 1
C      READ(10,*,END=80) N,(Y(I),I=1,N),(X(1,J),J=1,N)
C      KL = N - 1
C      WRITE(*,210)
C      WRITE(20,215)
C      WRITE(*,310)
C      WRITE(*,*)' (MAXIMUM value of K = 10)'
C      WRITE(*,*)
50 READ(*,*) W,KT
C      IF(KT .GT. KL) THEN
C          WRITE(*,320) KL
C          GOTO 50
C      ENDIF
55 DO 70 J=1,K
C          DO 60 L1=1,N
C              XJ = FLOAT(J)
C              X1(J,L1) = SIN(W*XJ*X(1,L1))
60      CONTINUE
70 CONTINUE
C      CALL REGMLT(X1,GAUSS)
C      WRITE(*,240) FI,FO,NN,(YHAT(I),Y(I),I=1,N)
C      WRITE(20,250)FI,FO,NN,(YHAT(I),Y(I),I=1,N)
C      WRITE(*,260) (B(I),I=1,K+1)
C      WRITE(20,260)(B(I),I=1,K+1)
C      WRITE(*,270) K,W,N,SDERR,PCERR,R
C      WRITE(20,270)K,W,N,SDERR,PCERR,R
C      IF(K .LT. KT) THEN
C          K = K + 1
C          PAUSE
C          GOTO 55
C      ENDIF
C      WRITE(*,280) NN
C      READ(*,120) T

```

```

      IF((T.EQ. 'N') .OR. (T.EQ. 'n')) GOTO 80
      NN = NN + 1
      GOTO 40
80  WRITE(*,290)
      WRITE(20,300)

C
C  *****  FORMAT STATEMENTS  *****
C
210 FORMAT(20(/)/2X,78('_')/2X,78('-')/28X,'TRIGONOMETRIC REGRE',
1      'SSION'/28X,24('-')/11X,'Y = B0 + B1.SIN(W.X) + ',
2      'B2.SIN(2W.X) ... + BK.SIN(K.W.X)'/)
215 FORMAT(/2X,78('_')/2X,78('-')/28X,'TRIGONOMETRIC REGRE',
1      'SSION'/28X,24('-')/11X,'Y = B0 + B1.SIN(W.X) + ',
2      'B2.SIN(2W.X) ... + BK.SIN(K.W.X)'/)
220 FORMAT(/2X,'X or Y can not be NEGATIVE.'/)
230 FORMAT(/2X,'Y can not be FRACTION.'/)
240 FORMAT(10(/)/2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,'/',
1      A18,3X,'DATA set #:',I3//24X,'Dependent variables (YHAT',
2      ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
3      E15.8,1X,E15.8))
250 FORMAT(/2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,'/',
1      A18,3X,'DATA set #:',I3//24X,'Dependent variables (YHAT',
2      ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
3      E15.8,1X,E15.8))
260 FORMAT(/4X,'REGRESSION coefficients'/4X,23('-')/
1      (1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8))
270 FORMAT(/2X,'No. of terms in each COS and/or SIN, [K] = ',I2,', '
1      ', and [W] = ',F8.4/2X,'No. of DATA points = ',I3,', SD',
2      'ERR = ',E15.8,', Percent ERROR = ',F8.4/2X,'Correlati',
3      'on coefficient, R = ',E15.8,'/')
280 FORMAT(/27X,'DATA set #: ',I3,' was read.'/10X,'If no more ',
1      'DATA to read Type [N], otherwise [Y] and <ENTER>.'/)
290 FORMAT(/////4X,73('#')/4X,'#',19X,'THANK you for using',
1      ' this package.',19X,'#'/4X,'#',71X,'#'/4X,'#',4X,
1      'REG2000 package has been developed by : Prof. Prabir ',
1      'K. Chandra',4X,'#'/4X,'# DCA/ESAL, 37200-000 Lavras, ',
1      'Minas Gerais, BRAZIL. FAX : (035)829-1100.#'/
1      4X,73('#')/10(/))
300 FORMAT(/2X,'SDERR: Standard ERROR = SQRT(((YHAT-Y)^2)/N).',
1      ' % ERROR = 100*SDERR/YM.'/2X,'YM: Mean of Y-',
2      'values, YHAT: Calculated VALUE of Y, R: Correlati',
3      'on coefficient.'/2X,78('_'))
310 FORMAT(/2X,'Give the value of W (REAL), value of K ',
1      '(INTEGER) and <ENTER>.'/)
320 FORMAT(/20X,'Insufficient DATA points. MAXIMUM K = ',I2/
1      17X,'Please again give values of W and K and <ENTER>')
      STOP
      END

C
$INCLUDE:'REGMLT.FOR'
$INCLUDE:'GAUSS.FOR'
C

```

A.3.5.8. Main program for Page's regression.

Program name : **MAIN.FOR**

```

C _____
C
C      Program developed by Prof. Prabir K. Chandra
C _____
C
C      PROGRAM MAIN
C
C      COMMON /CONS/ N,K,MM,L,KT
C      COMMON /BLK1/ A(12,12),Y(100),YHAT(100),B(12),G(12),YSSQ,SDERR,
C      I          PCERR,R
C      COMMON /BLK2/ X(12,100),X1(12,100)
C      COMMON /BLK3/ LL,L1,NN,JJ,M,W
C      CHARACTER *18 FI,FO
C      CHARACTER *1 T,Q
C      EXTERNAL GAUSS,REGMLT
C
C      ##### DATA INPUT #####
C
C $INCLUDE:'INCLUDE'
C
C      NN = 1
C      WRITE(20,210)
C      K = 1
50 READ(10,*,END=80) N,(Y(I),I=1,N),(X(1,J),J=1,N)
C      DO 60 J=1,N
C          IF(X(1,J) .EQ. 0.0) X(1,J) = 1.0E-05
C          IF((X(1,J) .LT. 0.0) .OR. (Y(J) .LT. 0.0)) THEN
C              WRITE(*,220)
C              PAUSE
C              NN = NN + 1
C              GOTO 50
C          ENDIF
C          IF(Y(J) .LT. 1.0) THEN
C              WRITE(*,230)
C              PAUSE
C              NN = NN + 1
C              GOTO 50
C          ENDIF
C          X(1,J) = ALOG(X(1,J))
C          Y(J) = ALOG(ALOG(Y(J)))
60 CONTINUE
C      CALL REGMLT(X,GAUSS)
C      B(1) = EXP(B(1))
C      DO 70 J=1,N
C          X(1,J) = EXP(X(1,J))
C          Y(J) = EXP(EXP(Y(J)))
C          YHAT(J) = EXP(EXP(YHAT(J)))
70 CONTINUE

```

```

WRITE(*,240) FI,FO,NN,(YHAT(I),Y(I),I=1,N)
WRITE(20,250)FI,FO,NN,(YHAT(I),Y(I),I=1,N)
WRITE(*,260) (B(I),I=1,K+1)
WRITE(20,260)(B(I),I=1,K+1)
WRITE(*,270) K,N,SDERR,PCERR,R
WRITE(20,270)K,N,SDERR,PCERR,R
WRITE(*,280) NN
READ(*,120) T
IF((T.EQ. 'N') .OR. (T.EQ. 'n')) GOTO 80
NN = NN + 1
GOTO 50
80 WRITE(*,290)
WRITE(20,300)

C
C ***** FORMAT STATEMENTS *****
C
210 FORMAT(2X,78('_')/2X,78('-')//32X,'PAGE-S REGRESSION'
1          /27X,26('-')//32X,'Y = e^(B0.X^B1)'/)
220 FORMAT(/2X,'X or Y can not be NEGATIVE.'/)
230 FORMAT(/2X,'Y can not be a FRACTION.'/)
240 FORMAT(10(/)2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,/,
1          A18,3X,'DATA set #:',I3//24X,'Dependent variables (YHAT',
2          ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
3          E15.8,1X,E15.8))
250 FORMAT(/2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,/,
1          A18,3X,'DATA set #:',I3//24X,'Dependent variables (YHAT',
2          ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
3          E15.8,1X,E15.8))
260 FORMAT(/4X,'REGRESSION coefficients'/4X,23('-')/
1          (1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8))
270 FORMAT(/2X,'No. of independent variables = ',I2,,' No. of,
1          ' DATA points =',I3/2X,'SDERR =',E15.8,,' Percent',
2          ' ERROR = ',F8.4,,' R =',E15.8,.'/)
280 FORMAT(/27X,'DATA set #: ',I3,' was read.'/10X,'If no more ',
1          'DATA to read Type [N], otherwise [Y] and <ENTER>.'/)
290 FORMAT(/////4X,73('#')/4X,'#',19X,'THANK you for using',
1          ' this package.',19X,'#'/4X,'#',71X,'#'/4X,'#',4X,
1          'REG2000 package has been developed by : Prof. Prabir ',
1          'K. Chandra',4X,'#'/4X,'# DCA/ESAL, 37200-000 Lavras, ',
1          'Minas Gerais, BRAZIL. FAX : (035)829-1100.#'/
1          4X,73('#')/10(/))
300 FORMAT(/2X,'SDERR: Standard ERROR = SQRT(((YHAT-Y)^2)/N).',
1          ' % ERROR = 100*SDERR/YM./2X,'YM: Mean of Y-',
2          'values, YHAT: Calculated VALUE of Y, R: Correla',
3          'tion coefficient.'/2X,78('_'))

STOP
END

C
$INCLUDE:'REGMLT.FOR'
$INCLUDE:'GAUSS.FOR'
C_____

```

A.3.5.9. Main program for asymptotic and logistic regressions.

Program name : **MAIN.FOR**

```

C _____
C
C      Program developed by Prof. Prabir K. Chandra
C _____
C
C      PROGRAM MAIN
C
C      COMMON /CONS/ N,K,MM,L,KT
C      COMMON /BLK1/ A(12,12),Y(100),YHAT(100),B(12),G(12),YSSQ,SDERR,
1      PCERR,R
C      COMMON /BLK2/ X(12,100),X1(12,100)
C      COMMON /BLK3/ LL,L1,NN,JJ,M,W
C      CHARACTER *18 FI,FO
C      CHARACTER *1 T,Q
C      EXTERNAL GAUSS,REGMLT
C
C      ##### DATA INPUT #####
C
C $INCLUDE:'INCLUDE'
C
C      NN = 1
C      K = 2
C      WRITE(20,210)
C      KT = 1
C      WRITE(*,220)
C      READ(*,*) M
C      IF((M.EQ. 0) .OR. (M .GE. 3)) THEN
C          REWIND 10
C          STOP
C      ENDIF
50 READ(10,*,END=80) N,(Y(I),I=1,N),(X(1,J),J=1,N)
C      CALL REGNON(1,M,GAUSS,REGMLT,R1)
60 WRITE(*,310) R1
C      READ(*,*) R1
C      CALL REGNON(2,M,GAUSS,REGMLT,R1)
C      IF(R1 .LT. 0.0) THEN
C          R1 = 0.99
C          WRITE(*,320) M,MM
C          READ(*,*) V
C          IF(V .GT. 1.0) THEN
C              NN = NN + 1
C              GOTO 50
C          ELSEIF((V .GT. 0.0) .AND. (V .LT. 1.0)) THEN
C              GOTO 60
C          ENDIF
C      ENDIF
C      WRITE(*,240) FI,FO,NN,(YHAT(I),Y(I),I=1,N)
C      WRITE(20,250) FI,FO,NN,(YHAT(I),Y(I),I=1,N)

```

```

WRITE(*,260) (B(I),I=1,K+1)
WRITE(20,260)(B(I),I=1,K+1)
WRITE(*,270) M,N,SDERR,PCERR,R,MM
WRITE(20,270)M,N,SDERR,PCERR,R,MM
WRITE(*,280) NN
READ(*,120) T
IF((T.EQ. 'N') .OR. (T.EQ. 'n')) GOTO 80
NN = NN + 1
GOTO 50
80 WRITE(*,290)
WRITE(20,300)

C
C ***** FORMAT STATEMENTS *****
C
210 FORMAT(2X,78('_')/2X,78('-')//30X,'NONLINEAR REGRESSION'/
1          30X,21('-')//15X,'Model [1]:  $Y = B_0 + B_1.e^{(B_2.X)}$  : ',
2          'Asymptotic'/15X,'Model [2]:  $Y = B_0/(1 + B_1.e^{(B_2.X)})$ ',
4          ': Logistic'/)
220 FORMAT(////////2X,78('_')/2X,78('-')//2X,'Type 1 or 2 and ',
1          '<ENTER>.'/2X,'1 : Asymptotic model:  $Y = B_0 + B_1.p^{X}$  ',
2          'or  $Y = B_0 + B_1.e^{(-B_2.X)}$ '/2X,'2 : Logistic model:  $Y =$ ',
3          ' $B_0/(1 + B_1.p^{X})$  or  $Y = B_0/((1 + B_1.e^{(-B_2.X)})/6X$ ,
4          'where,  $p = e^{(-B_2)}$ '/2X,'Any other DIGIT : MAIN ',
5          'OPTION MENU.'/2X,78('_')////)
240 FORMAT(10(/)2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,'/',
1          A18,3X,'DATA set #:',I3//24X,'Dependent variables (YHAT',
2          ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
3          E15.8,1X,E15.8))
250 FORMAT(2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,'/',
1          A18,3X,'DATA set #:',I3//24X,'Dependent variables (YHAT',
2          ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
3          E15.8,1X,E15.8))
260 FORMAT(4X,'REGRESSION coefficients'/4X,23('-')/
1          (1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8))
270 FORMAT(//2X,'MODEL #:',I2,'. No. of DATA points =',I3/
1          2X,'SDERR =',E15.8,'. Percent ERROR =',F8.4,'. R= ',
2          E15.8//2X,'No. of iterations =',I3,'.'/)
280 FORMAT(//27X,'DATA set #: ',I3,' was read.'/10X,'If no more ',
1          'DATA to read Type [N], otherwise [Y] and <ENTER>.'/)
290 FORMAT(////////4X,73('#')/4X,'#',19X,'THANK you for using',
1          ' this package.'/19X,'#'/4X,'#',71X,'#'/4X,'#',4X,
1          'REG2000 package has been developed by : Prof. Prabir ',
1          'K. Chandra',4X,'#'/4X,'# DCA/ESAL, 37200-000 Lavras, ',
1          'Minas Gerais, BRAZIL. FAX : (035)829-1100.#'/
1          4X,73('#')/10(/))
300 FORMAT(//2X,'SDERR: Standard ERROR =  $\sqrt{((YHAT-Y)^2)/N}$ :',
1          ' % ERROR =  $100*SDERR/YM$ .'/2X,'YM: Mean of Y-',
2          'values, YHAT: Calculated VALUE of Y, R: Correla',
3          'tion coefficient.'/2X,78('_'))
310 FORMAT(//2X,'Give the first assumption of p [=',F7.4,
1          ']' and <ENTER>.'/)
320 FORMAT(//2X,'Model #',I2,'. No. of iterations =',I3
1          /2X,'The scheme is NOT converging. Give a VALUE ',

```



```

2          '(0<VALUE<1) to continue'/2X,'or give a VALUE greater',
3          ' than ONE to READ the next DATA set.'/)
      STOP
      END
C
$INCLUDE:'REGMLT.FOR'
$INCLUDE:'GAUSS.FOR'
$INCLUDE:'REGNON.FOR'
C_____

```

A.3.5.10. Main program for polynomial regression.

Program name : **MAIN.FOR**

```

C_____
C
C      Program developed by Prof. Prabir K. Chandra
C_____
C
C      PROGRAM MAIN
C
C      COMMON /CONS/ N,K,MM,L,KT
C      COMMON /BLK1/ A(12,12),Y(100),YHAT(100),B(12),G(12),YSSQ,SDERR,
1      PCERR,R
C      COMMON /BLK2/ X(12,100),X1(12,100)
C      COMMON /BLK3/ LL,L1,NN,JJ,M,W
C      CHARACTER *18 FI,FO
C      CHARACTER *1 T,Q
C      EXTERNAL GAUSS,REGMLT
C
C      ##### DATA INPUT #####
C
C      $INCLUDE:'INCLUDE'
C
C      NN = 1
C      40 K = 2
C      READ(10,*,END=100) N,(Y(I),I=1,N),(X(1,J),J=1,N)
C      KL = N - 1
C      WRITE(20,210)
C      WRITE(*,220)
C      50 READ(*,*) KT
C      IF(KT .GT. KL) THEN
C      WRITE(*,310) KL
C      GOTO 50
C      ENDIF
C
C      THE MAXIMUM VALUE OF X-ARRAY IS SORTED
C
C      XMAX = -1.0E10
C      DO 60 J=1,N
C      XMAX = AMAX1(X(1,J),XMAX)

```

```

60 CONTINUE
C
C  NOW X-ARRAY IS DIVIDED BY XMAX
C
      DO 65 J=1,N
        X(1,J) = X(1,J)/XMAX
65 CONTINUE
C
70 DO 90 J=1,K
      DO 80 L1=1,N
        X(J,L1) = X(1,L1)**J
80 CONTINUE
90 CONTINUE
C
      CALL REGMLT(X,GAUSS)
C
C  REGRESSION COEFFICIENTS ARE BROUGHT TO THEIR ORIGINAL VALUES
C
      DO 95 I=2,K+1
        B(I) = B(I)/(XMAX**(I-1))
95 CONTINUE
C
      WRITE(*,240) FI,FO,NN,(YHAT(I),Y(I),I=1,N)
      WRITE(20,250)FI,FO,NN,(YHAT(I),Y(I),I=1,N)
      WRITE(*,260) (B(I),I=1,K+1)
      WRITE(20,260)(B(I),I=1,K+1)
      WRITE(*,270) K,N,SDERR,PCERR,R
      WRITE(20,270)K,N,SDERR,PCERR,R
      IF(K .LT. KT) THEN
        K = K + 1
        PAUSE
        GOTO 70
      ENDIF
      WRITE(*,280) NN
      READ(*,120) T
      IF((T .EQ. 'N') .OR. (T .EQ. 'n')) GOTO 100
      NN = NN + 1
      GOTO 40
100 WRITE(*,290)
      WRITE(20,300)
C
C  *****  FORMAT STATEMENTS  *****
C
210 FORMAT(2X,78('_')/2X,78('-')//30X,'POLYNOMIAL REGRESSION'/
1      30X,21('-')//27X,'Y = B0 + B1.X + B2.X^2 + ...'/)
220 FORMAT(/15X,'Give the degree of polynomial (MAXIMUM is 10).//)
240 FORMAT(10(/)/2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,'/',
1      A18,3X,'DATA set #:',I3//24X,'Dependent variables (YHAT',
2      ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,
3      E15.8,1X,E15.8))
250 FORMAT(/2X,78('_')/2X,'INPUT/OUTPUT file name: ',A18,'/',
1      A18,3X,'DATA set #:',I3//24X,'Dependent variables (YHAT',
2      ' and Y)/24X,32('-')/(1X,E15.8,1X,E15.8,1X,E15.8,1X,

```

```

3          E15.8,1X,E15.8))
260 FORMAT(/4X,'REGRESSION coefficients'/4X,23('-')/
1          (1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8,1X,E15.8))
270 FORMAT(/2X,'Degree of polynomial =',I2,'. No. of DATA ',
1          'points =',I3/2X,'SDERR =',E15.8,'. Percent ERROR ',
2          '= ',F8.4,'. R =',E15.8,'./)
280 FORMAT(/27X,'DATA set #: ',I3,' was read.'/10X,'If no more ',
1          'DATA to read Type [N], otherwise [Y] and <ENTER>.'/)
290 FORMAT(/4X,73('#')/4X,'#',19X,'THANK you for using',
1          ' this package.',19X,'#'/4X,'#',71X,'#'/4X,'#',4X,
1          'REG2000 package has been developed by : Prof. Prabir ',
1          'K. Chandra',4X,'#'/4X,'# DCA/ESAL, 37200-000 Lavras, ',
1          'Minas Gerais, BRAZIL. FAX : (035)829-1100.#'/
1          4X,73('#')/10(/))
300 FORMAT(/2X,'SDERR: Standard ERROR = SQRT(((YHAT-Y)^2)/N).',
1          ' % ERROR = 100*SDERR/YM.'/2X,'YM: Mean of Y-',
2          'values, YHAT: Calculated VALUE of Y, R: Correla',
3          'tion coefficient.'/2X,78('_'))
310 FORMAT(/28X,'Insufficient DATA points.'/8X,'Please again ',
1          'enter a degree LESS THAN or EQUAL TO ',I2,' and ',
2          '<ENTER>.'/)
      STOP
      END
C
$INCLUDE:'REGMLT.FOR'
$INCLUDE:'GAUSS.FOR'
C

```

APPENDIX A.3.6. Program listings for solutions of ODE.

A.3.6.1. Complete program listing for solving first-order ODE by various methods, such as Euler, Improved Euler, Runge-Kutta, Adams P-C and Hamming's P-C.

Program name : **ODE.FOR**

```
C-----
C  THIS PROGRAM COUPLED WITH THE SUBROUTINE 'ODE' CAN SOLVE ANY
C  FIRST ORDER ORDINARY DIFFERENTIAL EQUATION EMPLOYING (I) EULER,
C  (II) IMPROVED EULER, (III) RUNGE-KUTTA, (IV) ADAMS P-C AND
C  (V) HAMMINGS P-C.
C
C          PROGRAM DEVELOPED BY: DR. PRABIR K. CHANDRA
C-----
C
C          COMMON /BLOK1/ Y(5),FY(5),TT,Y0,FVALUE,DELT,ER1,L,K
C          CHARACTER *23 S
C          DOUBLE PRECISION Y,FY,TT,Y0,TIVALU,FVALUE,DELT,ER1
C          DOUBLE PRECISION F
C          EXTERNAL F
C
C          OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C          THE INITIAL CONDITION IS SPECIFIED
C
C          Y0 = 1.0
C
C          KK : INDEX NUMBER [1] for EULER; [2] for IMPROVED EULER
C              [3] for R-K; [4] for ADAMS; [5] HAMMINGS and [6] for
C              THE P-C METHOD DEVELOPED IN THIS TEXT.
C          DELT, INTV : STEPSIZE AND NO. OF VALUES TO BE SKIPPED.
C          TIVALU, FVALUE : INITIAL & FINAL VALUES OF T (IND. VARIABLE).
C
C          DO 20 II=1,100
C              WRITE(*,30)
C              READ(*,*) KK
C              IF(KK .EQ. 1) THEN
C                  S = 'E U L E R   M E T H O D'
C              ELSEIF(KK .EQ. 2) THEN
C                  S = 'IMPROVED EULER METHOD'
C              ELSEIF(KK .EQ. 3) THEN
C                  S = 'RUNGE-KUTTA M E T H O D'
```

```

ELSEIF(KK.EQ. 4) THEN
    S = 'ADAMS P-C METHOD'
ELSEIF(KK.EQ. 5) THEN
    S = 'HAMMINGS P-C METHOD'
ELSE
    CLOSE(20,STATUS='KEEP')
    STOP
ENDIF
WRITE(*,40)
READ(*,*) INTV,TIVALU,FVALUE,DELT

C
C
C
STARTING POINT IS INITIALIZED

K = 1
IK = 0
TT = TIVALU
ER1 = 0.0

C
C
C
INITIAL CONDITION IS ATTRIBUTED TO ACTUAL VARIABLE.

Y(1) = Y0
FY(1) = F(T,Y0)

C
WRITE(*,50) S
WRITE(20,60) S
WRITE(*,70) Y(1),TT
WRITE(20,70) Y(1),TT

C
C
C
NUMERICAL METHOD BEGINS BY CALLING SUBROUTINE 'ODE'

DO 10 J=2,500
    IK = IK + 1
    CALL ODE(KK,J,F)
    TT = TT + DELT

C
    IK1 = (IK/INTV)*INTV
    IF(IK1.EQ. IK) THEN
        WRITE(*,70) Y(L),TT
        WRITE(20,70) Y(L),TT
        IF(TT.LT. FVALUE) THEN
            PAUSE ''
        ELSE
            IF(KK.GE. 4) THEN
                WRITE(*,80) K,ER1,DELT

```

```

WRITE(20,80) K,ER1,DELT
ENDIF
WRITE(*,*)
PAUSE 'Going to the MAIN MENU'
GOTO 20
ENDIF
ENDIF
K = K + 1
10    CONTINUE
20    CONTINUE
C
C ***** FORMAT STATEMENTS *****
C
30    FORMAT(15(/)1X,79('#')/1X,'#',77X,'#/1X,'#',16X,'SOLUTION OF ',
1      'FIRST-ORDER DIFFERENTIAL EQUATION',16X,'#/1X,'#',77X,
1      '#/1X,'#',30X,'MAIN OPTION MENU',31X,'#/1X,'#',77X,'#'
1      '/1X,'#',25X,'CHOOSE ANY METHOD BY NUMBER',25X,'#/1X,'#',
1      '25X,27('-',)25X,'#/1X,'#',77X,'#/1X,'#',30X,'1: EULER',
1      '39X,'#/1X,'#',30X,'2: IMPROVED EULER',30X,'#/1X,'#',
1      '30X,'3: RUNGE-KUTTA',33X,'#/1X,'#',30X,'4: ADAMS P-C',
1      '35X,'#/1X,'#',30X,'5: HAMMINGS P-C',32X,'#/1X,'#',77X,
1      '#/1X,'#',23X,['Type any OTHER number to QUIT'],23X,
1      '#/1X,'#',77X,'#/1X,79('#')//)
40    FORMAT(20(/)4X,72('_')//12X,'TYPE in PRINTING INTERVAL (No. of ',
1      'VALUES to be skipped)/5X,'INITIAL & FINAL values of ',
2      'independent variable and STEP size - <ENTER>/4X,72('_')
3      //)
50    FORMAT(/28X,A23/28X,23('-')//20X,'SOLUTION Y and correspon',
1      'ding STEP values'/20X,40('-')/)
60    FORMAT(/28X,A23/28X,23('-')//20X,'SOLUTION Y and corresponding ',
1      'STEP values'/20X,40('-')/)
70    FORMAT(30X,D14.8,F6.2)
80    FORMAT(20X,40('-')//2X,'In ',I8,' steps, truncation error = ',
1      'D14.8,' with STEPSIZE = ',F6.4/)
STOP
END
C
C
C THIS SUBROUTINE EMPLOYS (I) EULER, (II) IMPROVED EULER, (III) 4-TH
C ORDER R-K METHOD, (IV) ADAMS P-C AND (V) HAMMINGS P-C.
C
C
C
SUBROUTINE ODE(KK,J,F)
COMMON /BLOK1/ Y(5),FY(5),TT,Y0,FVALUE,DELT,ER1,L,K

```

```

DOUBLE PRECISION Y,FY,TT,Y0,TIVALU,FVALUE,DELT,ER1
DOUBLE PRECISION YS(5),DY1,DY2,DY3,DY4,T,T1,TT1,ER,Y1,YY
DOUBLE PRECISION F

C
  IF(KK .EQ. 1) THEN
C
C      EULER METHOD STARTS ...
C
      L = 2
      T = TT
      YY = Y(1)
      Y(L) = Y(1) + DELT*F(T,YY)
      Y(1) = Y(L)
      RETURN
  ENDIF

C
  IF(KK .EQ. 2) THEN
C
C      IMPROVED EULER METHOD BEGINS ...
C
      L = 2
      T = TT
      YY = Y(1)
      Y1 = Y(1) + DELT*F(T,YY)
      Y(L) = Y(1) + DELT*(F(T,YY) + F(T+DELT,Y1))/2.0
      Y(1) = Y(L)
      RETURN
  ENDIF

C
  IF((J .LE. 4) .OR. (KK .EQ. 3)) THEN
C
C      STEPS ARE COMPUTED BY THE 4-TH ORDER RUNGE-KUTTA METHOD C
      L = 1
      T = TT
      YY = Y(1)
      DY1 = DELT*F(T,YY)
      T = TT + DELT/2.0
      YY = Y(1) + DY1/2.0
      DY2 = DELT*F(T,YY)
      YY = Y(1) + DY2/2.0
      DY3 = DELT*F(T,YY)
      T = TT + DELT
      YY = Y(1) + DY3
      DY4 = DELT*F(T,YY)

```

```

C
      Y(L) = Y(L) + (DY1 + 2.0*DY2 + 2.0*DY3 + DY4)/6.0
      YY = Y(L)
      IF(J .LE. 4) THEN
            Y(J) = YY
            FY(J) = F(T,YY)
            L = J
      ENDIF
      RETURN
ENDIF

C
      IF((J .EQ. 5) .AND. (KK .GT. 3)) THEN
            Y(1) = Y0
      ENDIF

C
      L = 5
      IF(K .GE. 5) THEN
            IF(KK .EQ. 4) THEN
                  XMOD = 251.0*(Y(L) - Y1)/270.0
            ELSEIF(KK .EQ. 5) THEN
                  XMOD = 112.0*(Y(L) - Y1)/121.0
            ENDIF
      ENDIF
      IF(KK .EQ. 4) THEN

C
C      ADAM'S PREDICTORS ARE CALCULATED...
C
      YS(L) = Y(4) + DELT*(55.0*FY(4)-59.0*FY(3)+37.0*FY(2)-
1      9.0*FY(1))/24.0
      ELSEIF(KK .EQ. 5) THEN

C
C      HAMMING'S PREDICTORS ARE CALCULATED...
C
      YS(L) = Y(1) + 4.0*DELT*(2.0*FY(4)-FY(3)+2.0*FY(2))/3.0
      ENDIF

C
C      THE PREDICTOR IS MODIFIED
C
      IF(K .GE. 5) THEN
            YS(L) = YS(L) + XMOD
      ENDIF
      Y1 = YS(L)
      FY(L) = F(TT,Y1)

C

```



```

ER = 1.0D-010
TT1 = 1.0D+010
DO 10 IJ=1,10
    IF(TT1 .LE. ER) GOTO 20
    IF(KK .EQ. 4) THEN
C
C   ADAM'S CORRECTORS ARE ITERATED TO CONVERGENCE...
C
        Y(L) = Y(4) + DELT*(9.0*FY(5)+19.0*FY(4)-5.0*FY(3)+
1          FY(2))/24.0
        ELSEIF(KK .EQ. 5) THEN
C
C   HAMMING'S CORRECTORS ARE ITERATED TO CONVERGENCE...
C
        Y(L) = (9.0*Y(4)-Y(2))/8.0 + 3.0*DELT*(FY(5)+2.0*FY(4)
1          -FY(3))/8.0
        ENDIF
        YY = Y(L)
        FY(L) = F(TT,YY)
        TT1 = DABS(Y(L) - YS(L))
        YS(L) = Y(L)
10    CONTINUE
C
C   THE TRUNCATION ERROR PER STEP IS SUMMED UP
C
20    IF(KK .EQ. 4) THEN
        ER1 = ER1 - 19.0*(YY - Y1)/270.0
    ELSEIF(KK .EQ. 5) THEN
        ER1 = ER1 + 9.0*(YY - Y1)/121.0
    ENDIF
C
C   ARRAYS ARE REARRANGED FOR THE NEXT STEP....
C
    DO 30 I=1,4
        Y(I) = Y(I+1)
        FY(I) = FY(I+1)
30    CONTINUE
C
    RETURN
    END
C
C
C   THIS FUNCTION DEFINES ALL THE FIRST-ORDER DIFFERENTIAL
C   EQUATIONS....

```

```

C
C
      DOUBLE PRECISION FUNCTION F(T,YY)
      COMMON /BLOK1/ Y(5),FY(5),TT,Y0,FVALUE,DELT,ER1,L,K
      DOUBLE PRECISION Y,FY,TT,Y0,FVALUE,DELT,ER1,T,YY
C
      F = - 2.0*T*YY
      RETURN
      END
C

```

A.3.6.2. Example PROGRAM of input DATA to N_ODE.FOR IN APPENDIX A.3.6.6.

Program name : **DATA.FOR**

```

C-----
C          ***** Program : DATA.FOR *****
C-----
C      ATTRIBUTE INITIAL VALUES TO THE VARIABLES Y0(1,1), Y0(1,2) ...,
C      Y0(1,N), Y0(2,1), Y0(2,2) ... , Y0(2,N) etc. IN CASE OF 'NEQ'
C      SETS OF Nth ORDER D.E. IN CASE OF A SET OF FIRST ORDER D.E.,
C      THESE SHOULD BE Y0(1,1), Y0(2,1), Y0(3,1) ... , Y0(N,1).
C
      Y0(1,1) = 1.0
      Y0(2,1) = -1.0
      Y0(3,1) = 2.0
      Y0(4,1) = -6.0
C

```

A.3.6.3. Example PROGRAM of FUNCTION input to N_ODE.FOR.

Program name : **FCTN.FOR**

```

C-----
C
C          ***** Program : FCTN.FOR *****
C-----
C
C      This program 'FCTN.FOR' provides all the first-order DE to
C      the SUBROUTINE RKPC. Y(1,KK,M), Y(2,KK,M) etc. are the solu-

```

C tions Y, Y1 etc., respectively of the differential eqns.

C

```

      IF(I .EQ. 1) THEN
        F = Y(2, KK, M)
      ELSEIF(I .EQ. 2) THEN
        F = Y(3, KK, M)
      ELSEIF(I .EQ. 3) THEN
        F = Y(4, KK, M)
      ELSEIF(I .EQ. 4) THEN
        F = -4.0*Y(1, KK, M)*Y(4, KK, M)
      ENDIF
      RETURN
      END
  
```

C _____

A.3.6.4. Subroutine START to provide starting values to RKPC.

Program name : **START.FOR**

C _____

C

C THIS SUBROUTINE PROVIDES STARTING VALUES TO "RKPC"

C _____

C

```

      SUBROUTINE START
      COMMON /BLOK3/ Y(10,5,4),Z(10,5,4),Y0(10,4),YT(10,101),TT,ER1
      COMMON /BLOK4/ FVALUE,TIVALU,DELT,NEQ,NORDER,K,L
      DOUBLE PRECISION Y,Z,Y0,YT,TT,ER1,FVALUE,TIVALU,DELT
  
```

C

C STARTING POINT IS INITIALIZED

C

```

      K = 1
      TT = TIVALU
      ER1 = 0.0
  
```

C

C INITIAL CONDITIONS ARE ATTRIBUTED TO ACTUAL VARIABLES

C

```

      DO 20 I=1,NEQ
        DO 10 M=1,NORDER
          Y(I,1,M) = Y0(I,M)
          Z(I,1,M) = Y0(I,M)
          YT(I,1) = Y0(I,M)
  
```

10 CONTINUE

20 CONTINUE

C

```

      RETURN
      END

```

C

A.3.6.5. Subroutine **RKPC** for solution of ODE by 1. fourth-order R-K, 2. Adams P-C and 3. Hamming's P-C methods.

Program name : **RKPC.FOR**

C

C

```

C      THIS SUBROUTINE SOLVES A SET OF SIMULTANEOUS FIRST- AND HIGHER
C      ORDER DIFFERENTIAL EQUATIONS BY 1. RUNGE-KUTTA, 2. ADAMS P-C
C      AND 3. HAMMING'S P-C METHODS.

```

C

```

C      PROGRAM DEVELOPED BY: DR. PRABIR K. CHANDRA

```

C

C

```

      SUBROUTINE RKPC(KK,J,F)
      COMMON /BLOK3/ Y(10,5,4),Z(10,5,4),Y0(10,4),YT(10,101),TT,ER1
      COMMON /BLOK4/ FVALUE,TIVALU,DELT,NEQ,NORDER,K,L
      DOUBLE PRECISION FY(10,5,4),ZS(10,5,4),XMOD(10,4),Y,Z,Y0,YT
      DOUBLE PRECISION Y1(10,4),DY1(10,4),DY2(10,4),DY3(10,4),DY4(10,4)
      DOUBLE PRECISION TT,ER1,FVALUE,TIVALU,DELT,T,T1,TT1,ER
      DOUBLE PRECISION F

```

C

```

C      RUNGE-KUTTA METHOD STARTS ...

```

C

```

      IF((J .LE. 4) .OR. (KK .EQ. 1)) THEN

```

```

          L = 1

```

C

```

          T = TT

```

```

          DO 20 I=1,NEQ

```

```

              DO 10 M=1,NORDER

```

```

                  DY1(I,M) = DELT*F(I,L,M,T)

```

```

10              CONTINUE

```

```

20          CONTINUE

```

C

```

          T = TT + DELT/2.0

```

```

          DO 40 I=1,NEQ

```

```

              DO 30 M=1,NORDER

```

```

                  Y(I,L,M) = Z(I,L,M) + DY1(I,M)/2.0

```

```

30              CONTINUE

```

```

40          CONTINUE

```

```

          DO 60 I=1,NEQ

```

```

                DO 50 M=1,NORDER
                DY2(I,M) = DELT*F(I,L,M,T)
50             CONTINUE
60             CONTINUE
C
                DO 80 I=1,NEQ
                DO 70 M=1,NORDER
                Y(I,L,M) = Z(I,L,M) + DY2(I,M)/2.0
70             CONTINUE
80             CONTINUE
                DO 100 I=1,NEQ
                DO 90 M=1,NORDER
                DY3(I,M) = DELT*F(I,L,M,T)
90             CONTINUE
100            CONTINUE
C
                T = TT + DELT
                DO 120 I=1,NEQ
                DO 110 M=1,NORDER
                Y(I,L,M) = Z(I,L,M) + DY3(I,M)
110            CONTINUE
120            CONTINUE
                DO 140 I=1,NEQ
                DO 130 M=1,NORDER
                DY4(I,M) = DELT*F(I,L,M,T)
130            CONTINUE
140            CONTINUE
C
                DO 160 I=1,NEQ
                DO 150 M=1,NORDER
                Z(I,L,M) = Z(I,L,M) + (DY1(I,M) + 2.0*DY2(I,M) +
1             2.0*DY3(I,M) + DY4(I,M))/6.0
                Y(I,L,M) = Z(I,L,M)
150            CONTINUE
160            CONTINUE
                IF(J .LE. 4) THEN
                DO 180 I=1,NEQ
                DO 170 M=1,NORDER
                Y(I,J,M) = Z(I,L,M)
                FY(I,J,M) = F(I,L,M,T)
170            CONTINUE
180            CONTINUE
                L = J
                ENDIF

```

```

        RETURN
    ENDIF

C
    IF((J .EQ. 5) .AND. (KK .GT. 1)) THEN
        DO 200 I=1,NEQ
            DO 190 M=1,NORDER
                Y(I,1,M) = Y0(I,M)
                Z(I,1,M) = Y0(I,M)
190          CONTINUE
200        CONTINUE
    ENDIF

C
C    ALL THE VALUES OF Y CALCULATED BY R-K METHOD ARE ASCERTAINED
C
    DO 230 L=2,4
        DO 220 I=1,NEQ
            DO 210 M=1,NORDER
                Z(I,L,M) = Y(I,L,M)
210          CONTINUE
220        CONTINUE
230      CONTINUE

C
    L = 5
    IF(K .GE. 5) THEN
        DO 250 I=1,NEQ
            DO 240 M=1,NORDER
                IF(KK .EQ. 2) THEN
                    XMOD(I,M) = 251.0*(Z(I,L,M) - Y1(I,M))/270.0
                ELSEIF(KK .EQ. 3) THEN
                    XMOD(I,M) = 112.0*(Z(I,L,M) - Y1(I,M))/121.0
                ENDIF
240          CONTINUE
250        CONTINUE
    ENDIF
    DO 270 I=1,NEQ
        DO 260 M=1,NORDER
            IF(KK .EQ. 2) THEN

C
C    ADAMS PREDICTOR IS CALCULATED ...
C
                ZS(I,5,M) = Z(I,4,M) + DELT*(55.0*FY(I,4,M)-59.0*
1              FY(I,3,M)+37.0*FY(I,2,M)-9.0*FY(I,1,M))/24.0
                ELSEIF(KK .EQ. 3) THEN

```

C HAMMING'S PREDICTOR IS CALCULATED ...

C

$$ZS(I,5,M) = Z(I,1,M) + 4.0*DELT*(2.0*FY(I,4,M)-FY(I,3,M) \\ +2.0*FY(I,2,M))/3.0$$

1

ENDIF

C

C THE PREDICTORS ARE MODIFIED

C

IF(K .GE. 5) THEN

$$ZS(I,5,M) = ZS(I,5,M) + XMOD(I,M)$$

ENDIF

$$Y1(I,M) = ZS(I,5,M)$$

$$Y(I,5,M) = ZS(I,5,M)$$

260 CONTINUE

270 CONTINUE

DO 290 I=1,NEQ

DO 280 M=1,NORDER

$$FY(I,5,M) = F(I,5,M,TT)$$

280 CONTINUE

290 CONTINUE

C

$$ER = 1.0D-10$$

$$TT1 = 1.0D+10$$

DO 340 IJ=1,10

IF(TT1 .LE. ER) GOTO 350

DO 310 I=1,NEQ

DO 300 M=1,NORDER

IF(KK .EQ. 2) THEN

C

C ADAMS CORRECTOR IS CALCULATED ...

C

$$Z(I,5,M) = Z(I,4,M) + DELT*(9.0*FY(I,5,M)+19.0*$$

1

$$FY(I,4,M)-5.0*FY(I,3,M)+FY(I,2,M))/24.0$$

ELSEIF(KK .EQ. 3) THEN

C

C HAMMING'S CORRECTOR IS CALCULATED ...

C

$$Z(I,5,M) = (9.0*Z(I,4,M)-Z(I,2,M))/8.0 + 3.0*DELT*$$

1

$$(FY(I,5,M)+2.0*FY(I,4,M)-FY(I,3,M))/8.0$$

ENDIF

$$Y(I,5,M) = Z(I,5,M)$$

300 CONTINUE

310 CONTINUE

DO 330 I=1,NEQ

DO 320 M=1,NORDER

```

                                FY(I,5,M) = F(I,5,M,TT)
320                                CONTINUE
330                                CONTINUE
                                TT1 = DABS(ZS(1,5,1)-Z(1,5,1))
                                ZS(1,5,1) = Z(1,5,1)
340                                CONTINUE
C
C   THE TRUNCATION ERROR ON Y PER STEP IS SUMMED UP
C
350                                IF(KK .EQ. 2) THEN
                                    ER1 = ER1 - 19.0*(Z(1,5,1) - Y1(1,1))/270.0
                                ELSEIF(KK .EQ. 3) THEN
                                    ER1 = ER1 + 9.0*(Z(1,5,1) - Y1(1,1))/121.0
                                ENDIF
C
C   ARRAYS ARE REARRANGED FOR THE NEXT STEP....
C
                                DO 380 JK=1,4
                                    DO 370 I=1,NEQ
                                        DO 360 M=1,NORDER
                                            Z(I,JK,M) = Z(I,JK+1,M)
                                            Y(I,JK,M) = Z(I,JK+1,M)
                                            FY(I,JK,M) = FY(I,JK+1,M)
360                                        CONTINUE
370                                    CONTINUE
380                                CONTINUE
C
                                RETURN
                                END
C

```

A.3.6.6. Complete PROGRAM for solution of ODE of any order by (1) R-K, (2) Adams P-C and (3) Hamming's P-C methods.

Program name : **N_ODE.FOR**

```

C
C   THIS PROGRAM COUPLED WITH THE SUBROUTINE 'RKPC' CAN SOLVE A
C   SET OF TEN (DEPENDING ON THE DIMENSION OF Y, ZS AND Z) FIRST
C   ORDER DIFFERENTIAL EQUATIONS SIMULTANEOUSLY OR 10-TH ORDER
C   ORDINARY DIFFERENTIAL EQUATION OR A SET OF TEN HIGHER ORDER
C   DIFFERENTIAL EQUATIONS (A MAXIMUM OF 4-TH ORDER) BY EMPLOYING
C   1. 4-TH ORDER RUNGE-KUTTA METHOD, 2. ADAM'S P-C METHOD AND
C   3. HAMMING'S P-C METHOD.

```



```

C
C      PROGRAM DEVELOPED BY ; DR. PRABIR K. CHANDRA
C
C
C      COMMON /BLOK3/ Y(10,5,4),Z(10,5,4),Y0(10,4),YT(10,101),TT,ERI
C      COMMON /BLOK4/ FVALUE,TIVALU,DELT,NEQ,NORDER,K,L
C      CHARACTER *35 S
C      DOUBLE PRECISION Y,Z,Y0,YT,TT,ERI,FVALUE,TIVALU,DELT
C      DOUBLE PRECISION F
C      EXTERNAL F
C
C      OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C      'DATA.FOR' IS A SEPARATE FILE TO BE CREATED WHERE THE INITIAL
C      CONDITIONS ARE SPECIFIED
C
C      $INCLUDE:'DATA.FOR'
C
C      KK : INDEX NUMBER - [1] FOR R-K; [2] FOR ADAMS P-C and
C      [3] FOR HAMMING'S P-C METHODS
C      NEQ, DELT : NUMBER OF EQUATIONS AND STEPSIZE.
C      NORDER : ORDER OF 'NEQ' DIFFERENTIAL EQUATIONS
C      TIVALU, FVALUE : INITIAL & FINAL VALUES OF INDEPENDENT VAR.
C
C      DO 20 II=1,100
C          WRITE(*,30)
C          READ(*,*) KK
C          IF(KK.EQ. 1) THEN
C              S = 'FOURTH-ORDER RUNGE-KUTTA METHOD'
C          ELSEIF(KK.EQ. 2) THEN
C              S = 'ADAMS PREDICTOR-CORRECTOR METHOD'
C          ELSEIF(KK.EQ. 3) THEN
C              S = 'HAMMING'S PREDICTOR-CORRECTOR METHOD'
C          ELSE
C              CLOSE(20,STATUS='KEEP')
C              STOP
C          ENDIF
C          WRITE(*,40)
C          READ(*,*) NEQ,NORDER,INTV,TIVALU,FVALUE,DELT
C
C      CALL START
C
C      NUMERICAL METHOD BEGINS BY CALLING SUBROUTINE 'RKPC'
C

```

```

      IK = 0
C
      WRITE(*,50) S
      WRITE(20,50) S
      WRITE(*,60) ((Y(I,I,M),M=1,NORDER),I=1,NEQ),TT
      WRITE(20,60) ((Y(I,I,M),M=1,NORDER),I=1,NEQ),TT
C
      DO 10 J=2,500
          IK = IK + 1
          CALL RKPC(KK,J,F)
          TT = TT + DELT
C
          IK1 = (IK/INTV)*INTV
          IF(IK1.EQ. IK) THEN
              WRITE(*,60) ((Y(I,I,M),M=1,NORDER),I=1,NEQ),TT
              WRITE(20,60) ((Y(I,I,M),M=1,NORDER),I=1,NEQ),TT
              IF(TT.LT. FVALUE) THEN
                  PAUSE ''
              ELSE
                  IF(KK.NE. 1) THEN
                      WRITE(*,70) K,ER1
                      WRITE(20,70) K,ER1
                  ENDIF
                  WRITE(*,*)
                  PAUSE 'Going to the MAIN MENU'
                  GOTO 20
              ENDIF
          ENDIF
          K = K + 1
10      CONTINUE
20      CONTINUE
C
C      ***** FORMAT STATEMENTS *****
C
30      FORMAT(15(/)1X,79('#')/1X,'#',77X,'#'/1X,'#',16X,'SOLUTION OF ',
1          'DIFFERENTIAL EQUATION OF ANY ORDER',15X,'#'/1X,'#',77X,
1          '#'/1X,'#',30X,'MAIN OPTION MENU',31X,'#'/1X,'#',77X,'#'
1          '/1X,'#',25X,'CHOOSE ANY METHOD BY NUMBER',25X,'#'/1X,'#',
1          25X,27('-'),25X,'#'/1X,'#',77X,'#'/1X,'#',30X,'1: RUNGE',
1          '-KUTTA',33X,'#'/1X,'#',30X,'2: ADAMS P-C',35X,'#'/1X,
1          '#',30X,'3: HAMMINGS P-C',32X,'#'/1X,'#',77X,'#'/1X,'#',
1          23X,['Type any OTHER number to QUIT'],23X,'#'/1X,'#',77X,
1          '#'/1X,79('#')/)
40      FORMAT(20(/)36X,'TYPE IN'//2X,75('-')/3X,'No. of EQUATIONS, ',

```

```

1      'ORDER, PRINTING INTERVAL (No. of VALUES to be SKIPPED),'
2      /4X,'INITIAL & FINAL values of independent variable and',
3      ' STEP size - <ENTER> '//10X,['ORDER of each of DIFFER',
4      'ENTIAL equations can not EXCEED 4]//2X,75('-')//)
50  FORMAT('//10X,'SOLUTIONS Y, Z1, Z2 ... and corresponding STEP ',
1      'VALUES'/10X,53('-')//20X,A35/20X,35('#')//)
60  FORMAT(1X,5(D14.8,1X))
70  FORMAT(1X,75('-')//2X,'TOTAL TRUNCATION ERROR IN ',I8,' STEPS',
1      ' = ',D14.8/)
      STOP
      END

```

```

C _____
C
C      'START.FOR' IS A SUBROUTINE LISTED IN APPENDIX A.3.6.4.
C
$INCLUDE:'START.FOR'
C _____
C
C      'RKPC.FOR' IS A SUBROUTINE LISTED IN APPENDIX A.3.6.5.
C
$INCLUDE:'RKPC.FOR'
C _____
C
C      FUNCTION 'FCTN.FOR' TO DEFINE ALL THE FIRST-ORDER ODEs ...
C      (LISTED IN APPENDIX A.3.6.3).
C _____
C
      DOUBLE PRECISION FUNCTION F(I,KK,M,T)
      COMMON /BLOK3/ Y(10,5,4),Z(10,5,4),Y0(10,4),YT(10,101),TT,ER1
      DOUBLE PRECISION Y,Z,Y0,YT,TT,ER1,T
C
$INCLUDE:'FCTN.FOR'
C _____

```

A.3.6.7. Main PROGRAM for solution of boundary value problem of up to 10-th order by (1) R-K, (2) Adams P-C and (3) Hamming's P-C methods.

Program name : **BOUND.FOR**

```

C _____
C      THIS PROGRAM COUPLED WITH THE SUBROUTINE 'RKPC' CAN SOLVE A

```

```

C   BOUNDARY VALUE PROBLEM CONSISTING OF UP TO TEN FIRST-ORDER
C   DIFFERENTIAL EQUATIONS SIMULTANEOUSLY OR UP TO 10-TH ORDER
C   ORDINARY DIFFERENTIAL EQUATION BY EMPLOYING 1. 4-TH ORDER
C   RUNGE-KUTTA 2. ADAM'S P-C AND 3. HAMMING'S P-C METHODS.
C
C
COMMON /BLOK3/ Y(10,5,4),Z(10,5,4),Y0(10,4),YT(10,101),XX,ERI
COMMON /BLOK4/ FVALUE,XIVALU,DELX,NEQ,NORDER,K,L
CHARACTER *35 S
DOUBLE PRECISION Y,Z,Y0,YT,H(101),U(2),Y1(2),FVALUE,XIVALU,
1      DELX,XBOUND,DELTA,XX,ERI
DOUBLE PRECISION F
EXTERNAL F
C
OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C   'DATA.FOR' IS A SEPARATE FILE TO BE CREATED WHERE THE INITIAL
C   CONDITIONS ARE SPECIFIED
C
$INCLUDE:'DATA.FOR'
C
NORDER = 1
C
C   KK : INDEX NUMBER - [1] FOR R-K; [2] FOR ADAMS P-C and
C   [3] FOR HAMMING'S P-C METHODS
C   NEQ, XBOUND : TOTAL NUMBER OF EQUATIONS AND BOUNDARY VALUE
C   NORDER : ORDER OF 'NEQ' DIFFERENTIAL EQUATIONS = 1
C   XIVALU, FVALUE : INITIAL & FINAL VALUES OF INDEPENDENT VAR.
C   INTV,DELX : PRINTING INTERVAL AND STEP SIZE
C
DO 70 II=1,100
    WRITE(*,80)
    READ(*,*) KK
    IF(KK.EQ. 1) THEN
        S = 'FOURTH-ORDER RUNGE-KUTTA METHOD'
    ELSEIF(KK.EQ. 2) THEN
        S = 'ADAMS PREDICTOR-CORRECTOR METHOD'
    ELSEIF(KK.EQ. 3) THEN
        S = 'HAMMING'S PREDICTOR-CORRECTOR METHOD'
    ELSE
        CLOSE(20,STATUS='KEEP')
        STOP
    ENDIF
    WRITE(*,90)

```

```

      READ(*,*) NEQ,XIVALU,FVALUE,XBOUND
      NDIV = 1 + (FVALUE - XIVALU)/0.1
      WRITE(*,100) NDIV
      READ(*,*) NDIV,INTV
      DELX = (FVALUE - XIVALU)/NDIV
C
C   TWO GUESSES OF SLOPE ARE MADE ...
C
      WRITE(*,110)
      READ(*,*) U(1),U(2)
C
C   SLOPE U(1) IS GIVEN AS INITIAL CONDITION
C
      Y0(NEQ,1) = U(1)
      DELTA = U(2) - U(1)
C
C   NUMERICAL METHOD BEGINS BY CALLING SUBROUTINE 'RKPC'
C
      CALL START
      DO 10 J=2,NDIV+1
          CALL RKPC(KK,J,F)
          XX = XX + DELX
          K = K + 1
10      CONTINUE
C
C   SLOPE U(2) IS GIVEN AS INITIAL CONDITION
C
      Y0(NEQ,1) = U(2)
      Y1(1) = Y(NEQ-1,L,1) - XBOUND
      DO 40 I=1,100
          ICOUNT = I
          CALL START
          IK = 0
          LL = 1
          DO 30 J=2,NDIV+1
              IK = IK + 1
              CALL RKPC(KK,J,F)
              XX = XX + DELX
              IK1 = (IK/INTV)*INTV
              IF(IK1 .EQ. IK) THEN
                  LL = LL + 1
                  DO 20 N=1,NEQ
                      YT(N,LL) = Y(N,L,1)
                      H(LL) = XX

```

```

20             CONTINUE
               ENDIF
               K = K + 1
30             CONTINUE
               Y1(2) = Y(NEQ-1,L,1) - XBOUND
               DELTA = - Y1(2)*DELTA/(Y1(2) - Y1(1))
               U(2) = U(2) + DELTA
               IF(DABS(Y1(2)) .LE. 1.0D-06) GOTO 50
               Y1(1) = Y1(2)
               Y0(NEQ,1) = U(2)
40             CONTINUE
C
50             WRITE(*,120) S,ICOUNT,DABS(Y1(2))
               WRITE(20,120) S,ICOUNT,DABS(Y1(2))
               DO 60 JJ=1,LL
                   WRITE(*,130) (YT(I,JJ),I=1,NEQ),H(JJ)
                   WRITE(20,130) (YT(I,JJ),I=1,NEQ),H(JJ)
60             CONTINUE
               PAUSE
70             CONTINUE
C
C ***** FORMAT STATEMENTS *****
C
80             FORMAT(15(/)1X,79('#')1X,'#',77X,'#/1X,'#',21X,'SOLUTION OF ',
1               'BOUNDARY VALUE PROBLEMS',21X,'#/1X,'#',77X,
1               '#/1X,'#',30X,'MAIN OPTION MENU',31X,'#/1X,'#',77X,'#
1               /1X,'#',25X,'CHOOSE ANY METHOD BY NUMBER',25X,'#/1X,'#',
1               25X,27('-'),25X,'#/1X,'#',77X,'#/1X,'#',30X,'1: RUNGE',
1               '-KUTTA',33X,'#/1X,'#',30X,'2: ADAMS P-C',35X,'#/1X,
1               '#',30X,'3: HAMMINGS P-C',32X,'#/1X,'#',77X,'#/1X,'#',
1               23X,['Type any OTHER number to QUIT'],23X,'#/1X,'#',77X,
1               '#/1X,79('#')//)
90             FORMAT(20(/)36X,'TYPE IN'//1X,77('-')/2X,'No. of EQUATIONS ',
1               '[INTEGER], INITIAL & FINAL values of independent ',
2               'variable',/18X,'and BOUNDARY value [REAL numbers] - ',
3               '<ENTER>./9X,['NUMBER of 1st-order DIFFERENTIAL ',
4               'equations can not EXCEED 10]/8X,64('-')//)
100            FORMAT(20(/)36X,'TYPE IN'//12X,55('-')/13X,'Number of DIVISIONS',
1               ' and PRINTING interval and <ENTER>'/14X,['NUMBER of ',
2               'values to be PRINTED can not SURPASS 100]/17X,
3               '[Number of DIVISIONS for DELX = 0.1 is: ',14,']/12X,
4               55('-')//)
110            FORMAT(/19X,'Give TWO guesses of slope U (REAL numbers)')
120            FORMAT(/10X,'SOLUTIONS Y, Z1, Z2 ... and corresponding STEP ',

```

```

1          'VALUES'/10X,53('-')//20X,A35/20X,35('#')//4X,'(Number',
2          ' of iterations is ',I4,' to yield an ERROR = ',D14.8,')'
3          /)
130  FORMAT(1X,5(D14.8,1X))
      STOP
      END

C _____
C
C  THIS SUBROUTINE PROVIDES STARTING VALUES TO "RKPC"
C      (APPENDIX: A.3.6.4)
C _____
C
$INCLUDE:'START.FOR'
C _____
C
C  THIS SUBROUTINE SOLVES A SET OF SIMULTANEOUS FIRST- AND HIGHER
C  ORDER DIFFERENTIAL EQUATIONS BY 1. 4-TH ORDER RUNGE-KUTTA,
C  2. ADAMS P-C, AND 3. HAMMING'S P-C METHODS (APPENDIX: A.3.6.5)
C _____
C
$INCLUDE:'RKPC.FOR'
C _____
C
C  FUNCTION FCTN.FOR TO DEFINE ALL THE FIRST-ORDER ODEs ....
C _____
C
      DOUBLE PRECISION FUNCTION F(I,KK,M,X)
      COMMON /BLOK3/ Y(10,5,4),Z(10,5,4),Y0(10,4),YT(10,101),TT,ER1
      DOUBLE PRECISION Y,Z,Y0,YT,TT,ER1,X

C
$INCLUDE:'FCTN.FOR'
C _____

```

Appendix A.3.6.8. Subroutine to calculate printing interval Program name : **PRNT.FOR**

```

C _____
C
C  SUBROUTINE TO CALCULATE PRINTING INTERVAL OF RESULTS
C _____
C
      SUBROUTINE PRNT(INTV1)

```

```

COMMON /BLOK1/ Y(10,5,4),Z(10,5,4),YT(10,101,4),Y0(10,4),H(101),
1          TT,ER1,INTV
COMMON /BLOK2/ FVALUE,TIVALU,DELT,NEQ,NVALUE,IK,NORDER,NN,K,L,KK
DOUBLE PRECISION Y,Z,YT,Y0,H,TT,TIVALU,DELT,ER1
C
C      VALUES OF Y, Y1, .... ARE STORED AT REQUIRED INTERVALS
C
      IF(INTV .EQ. 1) THEN
          IK = IK + 1
          DO 20 I=1,NEQ
              DO 10 M=1,NORDER
                  YT(I,IK,M) = Z(I,L,M)
10          CONTINUE
20          CONTINUE
          H(IK) = TT
          ELSE
          IPRINT = INTV1 + 1
          NN1 = NN + INTV/2
          IF((K .EQ. IPRINT) .AND. (K .LE. NN1)) THEN
              IK = IK + 1
              DO 40 I=1,NEQ
                  DO 30 M=1,NORDER
                      YT(I,IK,M) = Z(I,L,M)
30          CONTINUE
40          CONTINUE
          H(IK) = TT
          INTV1 = IK*INTV
      ENDIF
  ENDIF
C
      RETURN
      END
C

```

Appendix A.3.6.9. Subroutine to provide starting values to "RKPC1.FOR"

Program name : **START1.FOR**

```

C
C
C      THIS SUBROUTINE PROVIDES STARTING VALUES TO "RKPC1"
C
C
SUBROUTINE START1

```

```

COMMON /BLOK1/ Y(10,5,4),Z(10,5,4),YT(10,101,4),Y0(10,4),H(101),
1      TT,ER1,INTV
COMMON /BLOK2/ FVALUE,TIVALU,DELT,NEQ,NVALUE,IK,NORDER,NN,K,L,KK
DOUBLE PRECISION Y,Z,YT,Y0,H,TT,ER1,FVALUE,TIVALU,DELT,T
C
C      STARTING POINT IS INITIALIZED
C
      TT = TIVALU
      H(1) = TIVALU
      T = TT
      ER1 = 0.0
C
C      PRINTING INTERVAL "INTV" IS CALCULATED
C
      NN = (FVALUE - TIVALU)/DELT
      INTV = NN/NVALUE
      TINT = FLOAT(NN/NVALUE)
      B = TINT - FLOAT(INTV)
      IF((B .GE. 0.5) .AND. (TINT .GT. 1.0)) THEN
          INTV = INTV+1
      ENDIF
      IF(TINT .LE. 1.0) THEN
          INTV = 1
      ENDIF
C
C      INITIAL CONDITIONS ARE ATTRIBUTED TO ACTUAL VARIABLES
C
      DO 20 I=1,NEQ
          DO 10 M=1,NORDER
              YT(I,1,M) = Y0(I,M)
              Z(I,1,M) = Y0(I,M)
              Y(I,1,M) = Y0(I,M)
          10      CONTINUE
      20      CONTINUE
C
      K = 1
      IK = 1
C
      RETURN
      END
C

```

Appendix A.3.6.10. Subroutine to solve ODE (initial and boundary value problems) by 1. 4-th order R.K., 2. Adam's PC, 3. Hamming's PC methods.

Program name : **RKPC1.FOR**

```

C
C
C   THIS SUBROUTINE SOLVES A SET OF SIMULTANEOUS FIRST- AND HIGHER
C   ORDER DIFFERENTIAL EQUATIONS BY 1. RUNGE-KUTTA, 2. ADAMS P-C
C   AND HAMMING'S P-C METHODS.
C
C
C   SUBROUTINE RKPC1(PRNT,F)
C   COMMON /BLOK1/ Y(10,5,4),Z(10,5,4),YT(10,101,4),Y0(10,4),H(101),
1      TT,ER1,INTV
C   COMMON /BLOK2/ FVALUE,TIVALU,DELT,NEQ,NVALUE,IK,NORDER,NN,K,L,KK
C   DOUBLE PRECISION Y,Z,YT,Y0,FY(10,5,4),ZS(10,5,4),XMOD(10,4),H
C   DOUBLE PRECISION DY1(10,4),DY2(10,4),DY3(10,4),DY4(10,4),Y1(10,4)
C   DOUBLE PRECISION FVALUE,TIVALU,DELT,TT,ER1,T,T1,TT1,ER
C   DOUBLE PRECISION F
C
C   RUNGE-KUTTA METHOD STARTS ...
C
C   INTV1 = INTV
C   DO 210 J=2,1000000
C       L = 1
C       T = TT
C       DO 20 I=1,NEQ
C           DO 10 M=1,NORDER
C               DY1(I,M) = DELT*F(I,L,M,T)
10          CONTINUE
20          CONTINUE
C
C       T = TT + DELT/2.0
C       DO 40 I=1,NEQ
C           DO 30 M=1,NORDER
C               Y(I,L,M) = Z(I,L,M) + DY1(I,M)/2.0
30          CONTINUE
40          CONTINUE
C       DO 60 I=1,NEQ
C           DO 50 M=1,NORDER
C               DY2(I,M) = DELT*F(I,L,M,T)
50          CONTINUE
60          CONTINUE

```

C

```

DO 80 I=1,NEQ
    DO 70 M=1,NORDER
        Y(I,L,M) = Z(I,L,M) + DY2(I,M)/2.0
70     CONTINUE
80     CONTINUE
    DO 100 I=1,NEQ
        DO 90 M=1,NORDER
            DY3(I,M) = DELT*F(I,L,M,T)
90     CONTINUE
100    CONTINUE

```

C

```

T = TT + DELT
DO 120 I=1,NEQ
    DO 110 M=1,NORDER
        Y(I,L,M) = Z(I,L,M) + DY3(I,M)
110    CONTINUE
120    CONTINUE
    DO 140 I=1,NEQ
        DO 130 M=1,NORDER
            DY4(I,M) = DELT*F(I,L,M,T)
130    CONTINUE
140    CONTINUE

```

C

```

DO 160 I=1,NEQ
    DO 150 M=1,NORDER
        Z(I,L,M) = Z(I,L,M) + (DY1(I,M) + 2.0*DY2(I,M) +
1      2.0*DY3(I,M) + DY4(I,M))/6.0
        Y(I,L,M) = Z(I,L,M)
150    CONTINUE
160    CONTINUE
    IF(J .LE. 4) THEN
        DO 180 I=1,NEQ
            DO 170 M=1,NORDER
                Y(I,J,M) = Z(I,L,M)
                FY(I,J,M) = F(I,L,M,T)
170        CONTINUE
180        CONTINUE
    ENDIF

```

C

C PRINTING COUNTER K IS INCREMENTED BY 1 AND THE STEP BY DELT

C

```

IF(K .GT. NN) RETURN
K = K + 1

```

```

      TT = TT + DELT
C
C   VALUES OF Y, Y1, .... ARE STORED AT REQUIRED INTERVALS
C
      CALL PRNT(INTV1)
C
      IF((J.EQ. 4) .AND. (KK.GT. 1)) THEN
        DO 200 I=1,NEQ
          DO 190 M=1,NORDER
            Y(I,L,M) = Y0(I,M)
            Z(I,L,M) = Y0(I,M)
190          CONTINUE
200          CONTINUE
            GOTO 220
          ENDIF
210  CONTINUE
C
C   ALL THE VALUES OF Y CALCULATED BY R-K METHOD ARE ASCERTAINED
C
220  DO 250 L=2,4
        DO 240 I=1,NEQ
          DO 230 M=1,NORDER
            Z(I,L,M) = Y(I,L,M)
230          CONTINUE
240          CONTINUE
250          CONTINUE
C
C   STEP TT IS INCREMENTED BY DELT AND COUNTER K BY UNITY
C
260  TT = TT + DELT
      K = K + 1
      L = 5
      IF(K.GE. 6) THEN
        DO 280 I=1,NEQ
          DO 270 M=1,NORDER
            IF(KK.EQ. 2) THEN
              XMOD(I,M) = 251.0*(Z(I,L,M) - Y1(I,M))/270.0
            ELSEIF(KK.EQ. 3) THEN
              XMOD(I,M) = 112.0*(Z(I,L,M) - Y1(I,M))/121.0
            ENDIF
270          CONTINUE
280          CONTINUE
        ENDIF
        DO 300 I=1,NEQ

```

```

DO 290 M=1,NORDER
      IF(KK .EQ. 2) THEN
C
C   ADAMS PREDICTOR IS CALCULATED ...
C
          ZS(I,5,M) = Z(I,4,M) + DELT*(55.0*FY(I,4,M)-59.0*
1              FY(I,3,M)+37.0*FY(I,2,M)-9.0*FY(I,1,M))/24.0
          ELSEIF(KK .EQ. 3) THEN
C
C   HAMMING'S PREDICTOR IS CALCULATED ...
C
          ZS(I,5,M) = Z(I,1,M) + 4.0*DELT*(2.0*FY(I,4,M)-FY(I,3,M)
1              +2.0*FY(I,2,M))/3.0
          ENDIF
C
C   THE PREDICTORS ARE MODIFIED
C
          IF(K .GE. 6) THEN
              ZS(I,5,M) = ZS(I,5,M) + XMOD(I,M)
          ENDIF
          Y1(I,M) = ZS(I,5,M)
          Y(I,5,M) = ZS(I,5,M)
290      CONTINUE
300      CONTINUE
          DO 320 I=1,NEQ
              DO 310 M=1,NORDER
                  FY(I,5,M) = F(I,5,M,TT)
310          CONTINUE
320      CONTINUE
C
          ER = 1.0D-10
          TT1 = 1.0D+10
          DO 370 IJ=1,10
              IF(TT1 .LE. ER) GOTO 380
              DO 340 I=1,NEQ
                  DO 330 M=1,NORDER
                      IF(KK .EQ. 2) THEN
C
C   ADAMS CORRECTOR IS CALCULATED ...
C
                          Z(I,5,M) = Z(I,4,M) + DELT*(9.0*FY(I,5,M)+19.0*
1                              FY(I,4,M)-5.0*FY(I,3,M)+FY(I,2,M))/24.0
                          ELSEIF(KK .EQ. 3) THEN

```

```

C   HAMMING'S CORRECTOR IS CALCULATED ...
C
      Z(I,5,M) = (9.0*Z(I,4,M)-Z(I,2,M))/8.0 + 3.0*DELT*
1      (FY(I,5,M)+2.0*FY(I,4,M)-FY(I,3,M))/8.0
      ENDIF
      Y(I,5,M) = Z(I,5,M)
330    CONTINUE
340    CONTINUE
      DO 360 I=1,NEQ
        DO 350 M=1,NORDER
          FY(I,5,M) = F(I,5,M,TT)
350      CONTINUE
360    CONTINUE
      TT1 = DABS(ZS(1,5,1)-Z(1,5,1))
      ZS(1,5,1) = Z(1,5,1)
370  CONTINUE
C
C   THE TRUNCATION ERROR ON Y PER STEP IS SUMMED UP
C
380  IF(KK.EQ. 2) THEN
      ER1 = ER1 - 19.0*(Z(1,5,1) - Y1(1,1))/270.0
    ELSEIF(KK.EQ. 3) THEN
      ER1 = ER1 + 9.0*(Z(1,5,1) - Y1(1,1))/121.0
    ENDIF
C
C   VALUES OF Y1, Y2, .... ARE STORED AT REQUIRED INTERVALS
C
      CALL PRNT(INTV1)
C
C   ARRAYS ARE REARRANGED FOR THE NEXT STEP....
C
      DO 410 JK=1,4
        DO 400 I=1,NEQ
          DO 390 M=1,NORDER
            Z(I,JK,M) = Z(I,JK+1,M)
            Y(I,JK,M) = Z(I,JK+1,M)
            FY(I,JK,M) = FY(I,JK+1,M)
390      CONTINUE
400    CONTINUE
410  CONTINUE
      IF(K.GT. NN) RETURN
      GOTO 260
      RETURN
      END

```

C

A.3.6.11. Main PROGRAM for solution of boundary value problem of up to 10-th order by (1) R-K, (2) Adams P-C and (3) Hamming's P-C methods.

Program name : **BOUND1.FOR**

C

C THIS PROGRAM COUPLED WITH THE SUBROUTINE 'RKPC1' CAN SOLVE A
C BOUNDARY VALUE PROBLEM CONSISTING OF UP TO TEN FIRST-ORDER
C DIFFERENTIAL EQUATIONS SIMULTANEOUSLY OR UP TO 10-TH ORDER
C ORDINARY DIFFERENTIAL EQUATION BY EMPLOYING 1. 4-TH ORDER
C RUNGE-KUTTA 2. ADAM'S P-C AND 3. HAMMING'S P-C METHODS.

C

C

```
COMMON /BLOK1/ Y(10,5,4),Z(10,5,4),YT(10,101,4),Y0(10,4),H(101),
1          XX,ER1,INTV
COMMON /BLOK2/ FVALUE,XIVALU,DELX,NEQ,NVALUE,IK,NORDER,NN,K,L,KK
CHARACTER *35 S
DOUBLE PRECISION Y,Z,YT,Y0,H,U(2),Y1(2),XX,ER1,FVALUE,XIVALU,
1          DELX,XBOUND,DELTA
DOUBLE PRECISION F
EXTERNAL PRNT,F
```

C

```
OPEN(20,FILE='DATA.OUT',STATUS='NEW')
```

C

C 'DATA.FOR' IS A SEPARATE FILE TO BE CREATED WHERE THE INITIAL
C CONDITIONS ARE SPECIFIED

C

```
$INCLUDE:'DATA.FOR'
```

C

```
NORDER = 1
```

C

C KK : INDEX NUMBER - [1] FOR R-K; [2] FOR ADAMS P-C and
C [3] FOR HAMMING'S P-C METHODS

C NEQ, XBOUND : TOTAL NUMBER OF EQUATIONS AND BOUNDARY VALUE

C NORDER : ORDER OF 'NEQ' DIFFERENTIAL EQUATIONS = 1

C XIVALU, FVALUE : INITIAL & FINAL VALUES OF INDEPENDENT VAR.

C NVALUE,DELX : NO. OF VALUES TO BE PRINTED AND STEP SIZE

C

```
DO 40 II=1,100
```

```
WRITE(*,50)
```

```
READ(*,*) KK
```

```

      IF(KK .EQ. 1) THEN
        S = 'FOURTH-ORDER RUNGE-KUTTA METHOD'
      ELSEIF(KK .EQ. 2) THEN
        S = 'ADAMS PREDICTOR-CORRECTOR METHOD'
      ELSEIF(KK .EQ. 3) THEN
        S = 'HAMMINGS PREDICTOR-CORRECTOR METHOD'
      ELSE
        CLOSE(20,STATUS='KEEP')
        STOP
      ENDIF
      WRITE(*,60)
      READ(*,*) NEQ,NVALUE,XIVALU,FVALUE,DELX,XBOUND
C
C      TWO GUESSES OF SLOPE ARE MADE ...
C
      WRITE(*,70)
      READ(*,*) U(1),U(2)
C
C      SLOPE U(1) IS GIVEN AS INITIAL CONDITION
C
      Y0(NEQ,1) = U(1)
      DELTA = U(2) - U(1)
C      NUMERICAL METHOD BEGINS BY CALLING SUBROUTINE 'RKPC'
C
      CALL START1
      CALL RKPC1(PRNT,F)
C
C      SLOPE U(2) IS GIVEN AS INITIAL CONDITION
C
      Y0(NEQ,1) = U(2)
      Y1(1) = YT(NEQ-1,IK,1) - XBOUND
      DO 10 I=1,100
        ICOUNT = I
        CALL START1
        CALL RKPC1(PRNT,F)
        Y1(2) = YT(NEQ-1,IK,1) - XBOUND
        DELTA = - Y1(2)*DELTA/(Y1(2) - Y1(1))
        U(2) = U(2) + DELTA
        IF(DABS(Y1(2)) .LE. 1.0D-06) GOTO 20
        Y1(1) = Y1(2)
        Y0(NEQ,1) = U(2)
10      CONTINUE
C
20      WRITE(*,80) S,ICOUNT,DABS(Y1(2))

```



```

WRITE(20,80) S,ICOUNT,DABS(Y1(2))
DO 30 J=1,IK
    WRITE(*,90) (YT(I,J,1),I=1,NEQ),H(J)
    WRITE(20,90) (YT(I,J,1),I=1,NEQ),H(J)
30    CONTINUE
    PAUSE
40    CONTINUE
C
C ***** FORMAT STATEMENTS *****
C
50    FORMAT(15(/)1X,79('#')/1X,'#',77X,'#/1X,'#',21X,'SOLUTION OF ',
1        'BOUNDARY VALUE PROBLEMS',21X,'#/1X,'#',77X,
1        '#/1X,'#',30X,'MAIN OPTION MENU',31X,'#/1X,'#',77X,'#
1        /1X,'#',25X,'CHOOSE ANY METHOD BY NUMBER',25X,'#/1X,'#',
1        25X,27('-'),25X,'#/1X,'#',77X,'#/1X,'#',30X,'1: RUNGE',
1        '- KUTTA',33X,'#/1X,'#',30X,'2: ADAMS P-C',35X,'#/1X,
1        '#',30X,'3: HAMMING'S P-C',32X,'#/1X,'#',77X,'#/1X,'#',
1        23X,'Type any OTHER number to QUIT',23X,'#/1X,'#',77X,
1        '#/1X,79('#')//)
60    FORMAT(20(/)36X,'TYPE IN//8X,64('-')/12X,'No. of EQUATIONS, ',
1        'No. of VALUES to be printed (Integers),/15X,'also ',
2        'INITIAL & FINAL values of independent variable,/13X,
3        'STEP size and BOUNDARY value (Real numbers) - <ENTER>.',
4        //9X,'[NUMBER of 1st-order DIFFERENTIAL equations can ',
5        'not EXCEED 10./13X,'NUMBER of values to be PRINTED ',
6        'can not SURPASS 100]/8X,64('-')//)
70    FORMAT(/19X,'Give TWO guesses of slope U (REAL numbers)/)
80    FORMAT(/10X,'SOLUTIONS Y, Z1, Z2 ... and corresponding STEP ',
1        'VALUES'/10X,53('-')/20X,A35/20X,35('#')/4X,'(Number',
2        ' of iterations is ',I4,' to yield an ERROR = ',D14.8,')
3        /)
90    FORMAT(1X,5(D14.8,1X))
    STOP
    END
C
C
C THIS SUBROUTINE PROVIDES STARTING VALUES TO RKPC1 ...
C
$INCLUDE:'START1.FOR'
C
C
C THIS SUBROUTINE CALCULATES PRINTING INTERVAL ...
C
C

```

```
$INCLUDE:'PRNT.FOR'
```

```
C
```

```
C
```

```
C   THIS SUBROUTINE SOLVES A SET OF SIMULTANEOUS FIRST- AND HIGHER
C   ORDER DIFFERENTIAL EQUATIONS BY 1. RUNGE-KUTTA, 2. ADAMS P-C
C   AND HAMMING'S P-C METHODS.
```

```
C
```

```
C
```

```
$INCLUDE:'RKPC1.FOR'
```

```
C
```

```
C
```

```
C   FUNCTION FCTN.FOR TO DEFINE ALL THE FIRST-ORDER ODES ....
```

```
C
```

```
C
```

```
      DOUBLE PRECISION FUNCTION F(I, KK, M, X)
```

```
      COMMON /BLOK1/ Y(10,5,4), Z(10,5,4), YT(10,101,4), Y0(10,4), H(101),
```

```
      1          TT, ER1, INTV
```

```
      DOUBLE PRECISION Y, Z, YT, Y0, H, TT, ER1, X
```

```
C
```

```
$INCLUDE:'FCTN.FOR'
```

```
C
```



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

APPENDIX A.3.7. Program listings for the solution of partial differential equation.

A.3.7.1. Program listing for the solution of partial differential equation by Explicit method.

Program name : **EXPLCT.FOR**

```
C
C
C THIS PROGRAM COUPLED WITH THE SUBROUTINE 'EXPLCT' CAN SOLVE A
C PARTIAL DIFFERENTIAL EQUATION WITH DERIVATIVE BOUNDARY CONDI-
C TION BY THE EXPLICIT SCHEME.
C
C
C Program Developed by: Prof. Prabir K. Chandra
C
C
C
C COMMON /BLOK1/ TIME,TTIME,DELT,DD,TC,T0
C COMMON /BLOK2/ V(101),T(2,101),AR(101),V0,X1,DELR,N
C COMMON /PROP/ RO,ALFA,CP,XK,H,PI
C DOUBLE PRECISION TIME,TTIME,DELR,DELT,DD,TC,RO,ALFA,CP,XK,H
C DOUBLE PRECISION V,T,AR,V0,X1,RR,T0,TMEAN,PI
C DOUBLE PRECISION BC
C CHARACTER S *15, S2 *53
C CHARACTER S1 *20, S0 *1
C
C OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C DELR, DELT, TIME,TTIME : STEP SIZE IN SPACE (m) AND TIME (s),
C VARIABLE AND TOTAL TIME OF EXPOSURE (s), RESPECTIVELY.
C N,T : NUMBER OF DIVISIONS AND DEPENDENT VARIABLE (C).
C RO, ALFA, CP, XK, H : DENSITY (kg/m^3), DIFFUSIVITY (m^2/s),
C SPECIFIC HEAT (J/kg.C), CONDUCTIVITY (W/m.C) AND HEAT
C TRANSFER COEFFICIENT (W/m^2.C).
C AR(J), V(J) : J-TH AREA PERPENDICULAR TO FLOW (m^2) AND
C SEGMENT VOLUME (m^3), RESPECTIVELY.
C TC,T0 : TEMPERATURE AT SIDE 1 AND SIDE 2 AND INITIAL (C).
C
C DEFINE THERMO-PHYSICAL PROPERTIES ...
C
C $INCLUDE:'THERMO.FOR'
C
C
C PI = 3.141592654
C ALFA = XK/RO/CP
C S = 'EXPLICIT SCHEME'
```

```

DO 60 II=1,100
    WRITE(*,70) S
    READ(*,80) S0
    IF((S0.EQ. 'N') .OR. (S0.EQ. 'n')) THEN
        CLOSE(20,STATUS='KEEP')
        STOP
    ENDIF
    WRITE(*,90) S
    READ(*,*) L
    IF(L.EQ. 1) THEN
        S1 = 'RECTANGULAR GEOMETRY'
    ELSEIF(L.EQ. 2) THEN
        S1 = 'CYLINDRICAL GEOMETRY'
    ELSEIF(L.EQ. 3) THEN
        S1 = 'SPHERICAL GEOMETRY'
    ELSE
        GOTO 60
    ENDIF
    S2 = 'SIDE 1 : CONSTANT TEMPERATURE; SIDE 2 : DERIVATIVE BC'
    WRITE(*,100) S2
    READ(*,*) T0,TC
    WRITE(*,110)
    READ(*,*) X1,TTIME,N
    DELR = X1/N

C
    CALL GEOMTY(L)

C
    DELT = 0.5*DELR*DELR/ALFA
    WRITE(*,120) DELT
    READ(*,*) DELT
    NTIME = TTIME/DELT

C
C   TIME IS INITIALIZED ... INITIAL CONDITION IS ATTRIBUTED
C
    TIME = 0.0
    DO 10 J=1,N+1
        T(1,J) = BC(0,(J-1)*DELR)
10    CONTINUE

C
    WRITE(20,130) S,S1,S2
    WRITE(*,140)
    READ(*,*) KR
    IF(KR.EQ. 1) THEN
        WRITE(*,150) NTIME

```

```

      READ(*,*) INTV
      ELSEIF(KR.EQ. 2) THEN
        INTV = NTIME
      ELSE
        STOP
      ENDIF
      WRITE(*,160) S,S1,S2
C
C  INITIAL VALUES ARE PRINTED ...
C
      TMEAN = 0.0
      DO 20 J=1,N
        TMEAN = TMEAN + V(J)*(T(1,J) + T(1,J+1))/2.0/V0
20    CONTINUE
      WRITE(*,170) TMEAN,TIME,(T(1,J),J=1,N+1)
      WRITE(20,170) TMEAN,TIME,(T(1,J),J=1,N+1)
C
      IK = 0
C
      T(1,1) = BC(2,TC)
      T(1,N+1) = BC(2,T(1,N))
C
C  TIME IS INCREMENTED BY DELT AND BOUNDARY VALUES ARE GIVEN ...
C
30    TIME = TIME + DELT
      IK = IK + 1
C
      CALL EXPLCT
C
      T(2,1) = BC(2,TC)
      T(2,N+1) = BC(2,T(2,N))
C
      TMEAN = 0.0
      DO 40 J=1,N
        TMEAN = TMEAN + V(J)*(T(2,J) + T(2,J+1))/2.0/V0
40    CONTINUE
C
C  OLD VALUES ARE GIVEN NEW VALUES FOR THE NEXT TIMESTEP ...
C
      DO 50 J=1,N+1
        T(1,J) = T(2,J)
50    CONTINUE
C
      IK1 = (IK/INTV)*INTV

```

```

        IF(IK1 .EQ. IK) THEN
            WRITE(*,170) TMEAN,TIME,(T(1,J),J=1,N+1)
            WRITE(20,170) TMEAN,TIME,(T(1,J),J=1,N+1)
            IF(TIME .LT. TTIME) THEN
                PAUSE ' '
            ELSE
                WRITE(*,*)
                PAUSE 'Going to the MAIN MENU ...'
                GOTO 60
            ENDIF
        ENDIF
        GOTO 30
60    CONTINUE
C
C    ***** FORMAT STATEMENTS *****
C
70    FORMAT(25(/)32X,A15/32X,15('-')/20X,39('-')/21X,'Press [Y]',
1        ' to CONTINUE and [N] to QUIT'/20X,39('-')//)
80    FORMAT(A1)
90    FORMAT(25(/)1X,79('#')/1X,'#',77X,'#'/1X,'#',18X,'Solution ',
1        'of Partial Differential Equation',18X,'#'/1X,'#',18X,
1        41('-'),18X,'#'/1X,'#',77X,'#'/1X,'#',31X,A15,31X,'#/
1        1X,'#',77X,'#'/1X,'#',24X,'DECLARE PRODUCT CONFIGURA',
1        'TION',24X,'#'/1X,'#',24X,29('-'),24X,'#'/1X,'#',77X,
1        '#'/1X,'#',31X,'1: RECTANGULAR',32X,'#'/1X,'#',31X,
1        '2: CYLINDRICAL',32X,'#'/1X,'#',31X,'3: SPHERICAL',34X,
1        '#'/1X,'#',77X,'#'/1X,'#',13X,'[Type any OTHER number',
1        ' to RETURN to the BEGINNING]',14X,'#'/1X,'#',77X,'#/
1        1X,79('#')//)
100   FORMAT(25(/)3X,74('-')/13X,A53/13X,53('-')/15X,'Give INITIAL',
1        ' and SIDE 1 temperatures and <ENTER>.'/3X,74('-')//)
110   FORMAT(25(/)1X,78('-')/2X,'Give THICKNESS/RADIUS in [m], ',
1        'TOTAL time [s], No. of DIVISIONS and <ENTER>.'/1X,
1        78('-')//)
120   FORMAT(25(/)11X,56('-')/23X,'Give TIME step in [s] and <ENTER>'
1        '/12X,'[TIME step should be LESS THAN or EQUAL TO ',F11.5,
1        ']/'11X,56('-')//)
130   FORMAT(/21X,A15,' - ',A20/21X,38('-')/12X,[' ',A53,']/')
140   FORMAT(25(/)19X,42('-')/20X,'Type [1] : to PRINT step-by-step',
1        ' RESULTS'/20X,'Type [2] : to PRINT final RESULTS only.'
1        '/20X,'Type [3] : to STOP'/19X,42('-')//)
150   FORMAT(25(/)5X,71('-')/6X,'Give PRINTING INTERVAL [No. of ',
1        'TIME steps to be skipped] and <ENTER>.'/25X,['It ',
1        'SHOULD be LESS than ',I5,']/5X,71('-')//)

```

```

160  FORMAT(20(/)21X,A15,' - ',A20/21X,38('-')/12X,'|',A53,'|')
170  FORMAT(/12X,'Mean Temperature = ',F11.5,' at Time = ',F11.5,
1      ' s'/12X,54('-')//6(2X,F11.5))
      STOP
      END

```

C

C

C THIS SUBROUTINE CALCULATES AREA AND VOLUME OF DIFFERENT SHAPES

C

C

```

SUBROUTINE GEOMTY(L)
COMMON /BLOK2/ V(101),T(2,101),AR(101),V0,X1,DELR,N
DOUBLE PRECISION V,T,AR,V0,X1,DELR,PI

```

C

```

PI = 3.141592654
IF(L .EQ. 1) THEN
    DO 10 J=1,N+1
        AR(J) = 1.0
        V(J) = 1.0
10    CONTINUE
    V0 = FLOAT(N)
ELSEIF(L .GT. 1) THEN
    DO 20 J=1,N+1
        IF(L .EQ. 2) THEN
            AR(J) = 2.0*PI*(X1 - (J-1)*DELR)
            V(J) = PI*DELR*(2.0*X1 - (2*J-1)*DELR)
            V0 = PI*X1**2
        ELSEIF(L .EQ. 3) THEN
            AR(J) = 4.0*PI*(X1 - (J-1)*DELR)**2
            V(J) = 4.0*PI*DELR*(DELR**2 + 3.0*(X1 -
1            (J-1)*DELR)*(X1 - J*DELR))/3.0
            V0 = 4.0*PI*X1**3/3.0
        ENDIF
20    CONTINUE
ENDIF

```

C

```

RETURN
END

```

C

C

C THIS SUBROUTINE USES EXPLICIT METHOD TO SOLVE A PARTIAL DIFFERENTIAL
C EQUATION IN 1-D.

C

C

```

SUBROUTINE EXPLCT

```



```

COMMON /BLOK1/ TIME,TTIME,DELT,DD,TC,T0
COMMON /BLOK2/ V(101),T(2,101),AR(101),V0,X1,DELR,N
COMMON /PROP/ RO,ALFA,CP,XK,H,PI
DOUBLE PRECISION TIME,TTIME,DELR,DELT,DD,TC,T0
DOUBLE PRECISION V,T,AR,V0,X1,RO,ALFA,CP,XK,H,E,PI

C
DO 10 J=2,N
    E = ALFA*DELT/(2.0*AR(J)*DELR**2)
    T(2,J) = T(1,J) + E*((T(1,J+1)-T(1,J))*(AR(J+1)+AR(J)) -
1      (T(1,J)-T(1,J-1))*(AR(J)+AR(J-1)))
10 CONTINUE

C
RETURN
END

C
C
C THIS FUNCTION DEFINES INITIAL AND BOUNDARY CONDITIONS ....
C      (I = 0 FOR INITIAL CONDITION)
C
C
C
C
C DOUBLE PRECISION FUNCTION BC(I,TX)
C COMMON /BLOK1/ TIME,TTIME,DELT,DD,TC1,T0
C COMMON /PROP/ RO,ALFA,CP,XK,H,PI
C DOUBLE PRECISION TIME,TTIME,DELT,DD,DD1,DD2,TC1,TC2,T0,TX
C DOUBLE PRECISION RO,ALFA,CP,XK,H,PI

C
$INCLUDE:'FCTN.FOR'
C

```

A.3.7.2. Listing of subroutine GEOMTY(L) used in programs listed in Appendices A.3.7.4 through A.3.7.7.

Program name : **GEOMTY.FOR**

```

C
C
C THIS SUBROUTINE CALCULATES AREA AND VOLUME OF DIFFERENT SHAPES
C
C
C SUBROUTINE GEOMTY(L)
C COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
1      TS(101),AR(101),V0,X1,DELR,N
C DOUBLE PRECISION A,B,D,R,V,T,TS,AR,V0,X1,DELR,PI

C

```

```

      PI = 3.141592654
      IF(L .EQ. 1) THEN
        DO 10 J=1,N+1
          AR(J) = 1.0
          V(J) = 1.0
10      CONTINUE
        V0 = FLOAT(N)
      ELSEIF(L .GT. 1) THEN
        DO 20 J=1,N+1
          IF(L .EQ. 2) THEN
            AR(J) = 2.0*PI*(X1 - (J-1)*DELR)
            V(J) = PI*DELR*(2.0*X1 - (2*J-1)*DELR)
            V0 = PI*X1**2
          ELSEIF(L .EQ. 3) THEN
            AR(J) = 4.0*PI*(X1 - (J-1)*DELR)**2
            V(J) = 4.0*PI*DELR*(DELR**2 + 3.0*(X1 -
1          (J-1)*DELR)*(X1 - J*DELR))/3.0
            V0 = 4.0*PI*X1**3/3.0
          ENDIF
20      CONTINUE
        ENDIF
      C
      RETURN
      END
C

```

A.3.7.3. Listing of subroutine TRIDGL(N) used in programs listed in Appendices A.3.7.3 through A.3.7.5.

Program name : **TRIDGL.FOR**

```

C
C
C   THIS SUBROUTINE SOLVES A 1-D PARABOLIC PDE USING TRIDIAGONAL MATRIX.
C
C
C
      SUBROUTINE TRIDGL(NN)
      COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
1      TS(101),AR(101),V0,X1,DELR,N
      DOUBLE PRECISION A,B,D,R,V,T,TS,AR,V0,X1,DELR,AA
C
      A(NN) = 0.0
      B(2) = 0.0
      B(NN) = B(NN)/D(NN)

```

```

R(NN) = R(N)/D(NN)
C
DO 10 I=3,NN
    J = NN - I + 3
    AA = 1.0/(D(J-1) - B(J)*A(J-1))
    B(J-1) = B(J-1)*AA
    R(J-1) = (R(J-1) - A(J-1)*R(J))*AA
10  CONTINUE
C
C  BACK SUBSTITUTION STARTS ...
C
    TS(2) = R(2)
    DO 20 J=3,NN
        TS(J) = R(J) - B(J)*TS(J-1)
20  CONTINUE
C
    RETURN
    END
C

```

A.3.7.4. Program listing for the solution of partial differential equation by Fully Implicit method.

Program name : **IMPLCT.FOR**

```

C
C  THIS PROGRAM COUPLED WITH THE SUBROUTINES 'COEFF' AND 'TRIDGL'
C  CAN SOLVE A PARTIAL DIFFERENTIAL EQUATION WITH DERIVATIVE
C  BOUNDARY CONDITION BY THE IMPLICIT SCHEME.
C
C  Program Developed by: Prof. Prabir K. Chandra
C
C
COMMON /BLOK1/ TIME,TTIME,DELT,DD,TC,T0
COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
1      TS(101),AR(101),V0,X1,DELR,N
COMMON /PROP/ RO,ALFA,CP,XK,H,PI
DOUBLE PRECISION TIME,TTIME,DELR,DELT,DD,TC,RO,ALFA,CP,XK,
1      H,V0,X1,RR,T0,TMEAN,PI
DOUBLE PRECISION A,B,D,R,V,T,TS,AR
DOUBLE PRECISION BC
CHARACTER S *15, S2 *53
CHARACTER S1 *20, S0 *1
C

```

```

OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C      DELR, DELT, TIME, TTIME : STEP SIZE IN SPACE (m) AND TIME (s),
C      VARIABLE AND TOTAL TIME OF EXPOSURE (s), RESPECTIVELY.
C      D, A, B and N: DIAGONAL, UPPER, LOWER ELEMENTS and ROW OR
C      COLUMN SIZE OF TRIANGULAR MATRIX.
C      R, T: ELEMENTS OF CONSTANT VECTOR AND SOLUTION VECTOR.
C      RO, ALFA, CP, XK, H : DENSITY (kg/m^3), DIFFUSIVITY (m^2/s),
C      SPECIFIC HEAT (J/kg.C), CONDUCTIVITY (W/m.C) AND HEAT
C      TRANSFER COEFFICIENT (W/m^2.C).
C      AR(J), V(J) : J-TH AREA PERPENDICULAR TO FLOW (m^2) AND
C      SEGMENT VOLUME (m^3), RESPECTIVELY.
C      TC,T0 : TEMPERATURE AT SIDE 1 AND SIDE 2 AND INITIAL (C).
C
C      DEFINE THERMO-PHYSICAL PROPERTIES ...
C
$INCLUDE:'THERMO.FOR'
C
PI = 3.141592654
ALFA = XK/RO/CP
S = 'IMPLICIT SCHEME'
DO 60 II=1,100
    WRITE(*,70) S
    READ(*,80) S0
    IF((S0 .EQ. 'N') .OR. (S0 .EQ. 'n')) THEN
        CLOSE(20,STATUS='KEEP')
        STOP
    ENDIF
    WRITE(*,90) S
    READ(*,*) L
    IF(L .EQ. 1) THEN
        S1 = 'RECTANGULAR GEOMETRY'
    ELSEIF(L .EQ. 2) THEN
        S1 = 'CYLINDRICAL GEOMETRY'
    ELSEIF(L .EQ. 3) THEN
        S1 = 'SPHERICAL GEOMETRY'
    ELSE
        GOTO 60
    ENDIF
    S2 = 'SIDE 1 : CONSTANT TEMPERATURE; SIDE 2 : DERIVATIVE BC'
    WRITE(*,100) S2
    READ(*,*) T0,TC
    WRITE(*,110)
    READ(*,*) X1,TTIME,N

```

```

                                DELR = X1/N
C
                                CALL GEOMTY(L)
C
                                DELT = 0.5*DELR*DELR/ALFA
                                WRITE(*,120) DELT
                                READ(*,*) DELT
                                NTIME = TTIME/DELT
                                DD = -DELR*DELR/ALFA/DELT
C
C      TIME IS INITIALIZED ... INITIAL CONDITION IS ATTRIBUTED
C
                                TIME = 0.0
                                DO 10 J=1,N+1
                                    T(1,J) = BC(0,(J-1)*DELR)
10      CONTINUE
C
                                WRITE(20,130) S,S1,S2
                                WRITE(*,140)
                                READ(*,*) KR
                                IF(KR.EQ. 1) THEN
                                    WRITE(*,150) NTIME
                                    READ(*,*) INTV
                                ELSEIF(KR.EQ. 2) THEN
                                    INTV = NTIME
                                ELSE
                                    STOP
                                ENDIF
                                WRITE(*,160) S,S1,S2
C
C      INITIAL VALUES ARE PRINTED ...
C
                                TMEAN = 0.0
                                DO 20 J=1,N
                                    TMEAN = TMEAN + V(J)*(T(1,J) + T(1,J+1))/2.0/V0
20      CONTINUE
                                WRITE(*,170) TMEAN,TIME,(T(1,J),J=1,N+1)
                                WRITE(20,170) TMEAN,TIME,(T(1,J),J=1,N+1)
C
                                IK = 0
C
                                T(1,1) = BC(2,TC)
C
                                T(1,N+1) = BC(2,T(1,N))
C
C      TIME IS INCREMENTED BY DELT AND BOUNDARY VALUES ARE GIVEN ...

```

```

C
30      TIME = TIME + DELT
      IK = IK + 1

C
      CALL COEFF
      CALL TRIDGL(N)

C
      TS(1) = BC(2,TC)
      TS(N+1) = BC(2,TS(N))

C
      TMEAN = 0.0
      DO 40 J=1,N
          TMEAN = TMEAN + V(J)*(TS(J) + TS(J+1))/2.0/V0
40      CONTINUE

C
C      OLD VALUES ARE GIVEN NEW VALUES FOR THE NEXT TIMESTEP ...
C
      DO 50 J=1,N+1
          T(1,J) = TS(J)
50      CONTINUE

C
      IK1 = (IK/INTV)*INTV
      IF(IK1 .EQ. IK) THEN
          WRITE(*,170) TMEAN,TIME,(T(1,J),J=1,N+1)
          WRITE(20,170) TMEAN,TIME,(T(1,J),J=1,N+1)
          IF(TIME .LT. TTIME) THEN
              PAUSE ' '
          ELSE
              WRITE(*,*)
              PAUSE 'Going to the MAIN MENU ...'
              GOTO 60
          ENDIF
      ENDIF
      GOTO 30
60      CONTINUE

C
C      ***** FORMAT STATEMENTS *****
C
70      FORMAT(25(/)32X,A15/32X,15('-')/20X,39('-')/21X,'Press [Y]',
1          ' to CONTINUE and [N] to QUIT'/20X,39('-')/)
80      FORMAT(A1)
90      FORMAT(25(/)1X,79('#')/1X,'#',77X,'#'/1X,'#',18X,'Solution ',
1          'of Partial Differential Equation',18X,'#'/1X,'#',18X,
1          41('-'),18X,'#'/1X,'#',77X,'#'/1X,'#',31X,A15,31X,'#/'

```

```

1      1X,'#',77X,'#/1X,'#',24X,'DECLARE PRODUCT CONFIGURA',
1      'TION',24X,'#/1X,'#',24X,29('-'),24X,'#/1X,'#',77X,
1      '#/1X,'#',31X,'1: RECTANGULAR',32X,'#/1X,'#',31X,
1      '2: CYLINDRICAL',32X,'#/1X,'#',31X,'3: SPHERICAL',34X,
1      '#/1X,'#',77X,'#/1X,'#',13X,['Type any OTHER number',
1      ' to RETURN to the BEGINNING'],14X,'#/1X,'#',77X,'#/'
1      1X,79('#')//
100  FORMAT(25(/)3X,74('-')/13X,A53/13X,53('-')/15X,'Give INITIAL',
1      ' and SIDE 1 temperatures and <ENTER>.'/3X,74('-')//)
110  FORMAT(25(/)1X,78('-')/2X,'Give THICKNESS/RADIUS in [m], ',
1      'TOTAL time [s], No. of DIVISIONS and <ENTER>.'/11X,
1      78('-')//)
120  FORMAT(25(/)18X,41('-')/23X,'Give TIME step in [s] and <ENTER>.'
1      /19X,['Lower limit of TIME step = ',F10.5,']'/18X,41('-')
1      //)
130  FORMAT(21X,A15,' - ',A20/21X,38('-')/12X,['A53,']//)
140  FORMAT(25(/)19X,42('-')/20X,'Type [1] : to PRINT step-by-step',
1      ' RESULTS'/20X,'Type [2] : to PRINT final RESULTS only.'
1      /20X,'Type [3] : to STOP'/19X,42('-')//)
150  FORMAT(25(/)5X,71('-')/6X,'Give PRINTING INTERVAL [No. of ',
1      'TIME steps to be skipped] and <ENTER>.'/25X,['It ',
1      'SHOULD be LESS than ',I5,']'/5X,71('-')//)
160  FORMAT(20(/)21X,A15,' - ',A20/21X,38('-')/12X,['A53,']//)
170  FORMAT(/12X,'Mean Temperature = ',F11.5,' at Time = ',F11.5,
1      ' s'/12X,54('-')/6(2X,F11.5))

      STOP
      END

```

```

C
C
C  THIS SUBROUTINE CALCULATES VOLUME AND AREA OF DIFFERENT SHAPES
C
C
$INCLUDE:'GEOMTY.FOR'
C
C
C  THIS SUBROUTINE SUPPLIES COEFFICIENT VALUES TO "TRIDGL" ...
C
C

```

```

SUBROUTINE COEFF
COMMON /BLOK1/ TIME,TTIME,DELT,DD,TC,T0
COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
1      TS(101),AR(101),V0,X1,DELR,N
DOUBLE PRECISION TIME,TTIME,DELR,DELT,DD,TC,T0,C1,C2,C3,CC0
DOUBLE PRECISION A,B,D,R,V,T,TS,AR,V0,X1,DELR,CC

```

```

C
C   ELEMENTS OF TRIDIAGONAL MATRIX AND CONSTANT VECTOR ARE DEFINED ...
C
      DO 10 J=2,N
          C1 = (AR(J) + AR(J-1))/2.0/AR(J)
          C3 = (AR(J+1) + AR(J))/2.0/AR(J)
          C2 = DD - C1 - C3
          B(J) = C1
          D(J) = C2
          IF(J .EQ. N) D(J) = C2 + C3
          A(J) = C3
          R(J) = DD*T(1,J)
10    CONTINUE
C
C   THE FIRST AND THE LAST ELEMENTS OF THE CONSTANT VECTOR ARE
C   CORRECTED ACCORDING TO DIFFERENT BOUNDARY CONDITIONS ...
C
      R(2) = R(2) - B(2)*TC
C
      RETURN
      END
C


---


C
C   SUBROUTINE TRIDGL SOLVES 1-D PARABOLIC PDE USING TRIDIAGONAL MATRIX.
C


---


C
$INCLUDE:'TRIDGL.FOR'
C


---


C
C   THIS FUNCTION DEFINES INITIAL AND BOUNDARY CONDITIONS ....
C   (I = 0 FOR INITIAL CONDITION)
C


---


C
      DOUBLE PRECISION FUNCTION BC(I,TX)
      COMMON /BLOK1/ TIME,TTIME,DELT,DD,TC1,T0
      COMMON /PROP/ RO,ALFA,CP,XK,H,PI
      DOUBLE PRECISION TIME,TTIME,DELT,DD,DD1,DD2,TC1,TC2,T0,TX
      DOUBLE PRECISION RO,ALFA,CP,XK,H,PI
C
$INCLUDE:'FCTN.FOR'
C


---



```


A.3.7.5. Program listing for the solution of partial differential equation by Crank-Nicolson method.

Program name : **CRANK.FOR**

```

C
C _____
C   THIS PROGRAM COUPLED WITH THE SUBROUTINES 'COEFF' AND 'TRIDGL'
C   CAN SOLVE A PARTIAL DIFFERENTIAL EQUATION WITH DERIVATIVE
C   BOUNDARY CONDITION BY THE CRANK-NICOLSON SCHEME.
C
C   Program Developed by: Prof. Prabir K. Chandra
C
C _____
C
C   COMMON /BLOK1/ TIME,TTIME,DELT,DD,TC,T0
C   COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
1      TS(101),AR(101),V0,X1,DELR,N
C   COMMON /PROP/ RO,ALFA,CP,XK,H,PI
C   DOUBLE PRECISION TIME,TTIME,DELR,DELT,DD,TC,RO,ALFA,CP,XK,
1      H,V0,X1,RR,T0,TMEAN,PI
C   DOUBLE PRECISION A,B,D,R,V,T,TS,AR
C   DOUBLE PRECISION BC
C   CHARACTER S *15, S2 *53
C   CHARACTER S1 *20, S0 *1
C
C   OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C   DELR, DELT, TIME,TTIME : STEP SIZE IN SPACE (m) AND TIME (s),
C   VARIABLE AND TOTAL TIME OF EXPOSURE (s), RESPECTIVELY.
C   D, A, B and N: DIAGONAL, UPPER, LOWER ELEMENTS and ROW OR
C   COLUMN SIZE OF TRIANGULAR MATRIX.
C   R, T: ELEMENTS OF CONSTANT VECTOR AND SOLUTION VECTOR.
C   RO, ALFA, CP, XK, H : DENSITY (kg/m^3), DIFFUSIVITY (m^2/s),
C   SPECIFIC HEAT (J/kg.C), CONDUCTIVITY (W/m.C) AND HEAT
C   TRANSFER COEFFICIENT (W/m^2.C).
C   AR(J), V(J) : J-TH AREA PERPENDICULAR TO FLOW (m^2) AND
C   SEGMENT VOLUME (m^3), RESPECTIVELY.
C   TC,T0 : TEMPERATURE AT SIDE 1 AND SIDE 2 AND INITIAL (C).
C
C   DEFINE THERMO-PHYSICAL PROPERTIES ...
C
C $INCLUDE:'THERMO.FOR'
C
C   PI = 3.141592654
C   ALFA = XK/RO/CP
C   S = 'CRANK-NICOLSON'

```

```

DO 60 II=1,100
    WRITE(*,70) S
    READ(*,80) S0
    IF((S0.EQ. 'N') .OR. (S0.EQ. 'n')) THEN
        CLOSE(20,STATUS='KEEP')
        STOP
    ENDIF
    WRITE(*,90) S
    READ(*,*) L
    IF(L.EQ. 1) THEN
        S1 = 'RECTANGULAR GEOMETRY'
    ELSEIF(L.EQ. 2) THEN
        S1 = 'CYLINDRICAL GEOMETRY'
    ELSEIF(L.EQ. 3) THEN
        S1 = 'SPHERICAL GEOMETRY'
    ELSE
        GOTO 60
    ENDIF
    S2 = 'SIDE 1 : CONSTANT TEMPERATURE; SIDE 2 : DERIVATIVE BC'
    WRITE(*,100) S2
    READ(*,*) T0,TC
    WRITE(*,110)
    READ(*,*) X1,TTIME,N
    DELR = X1/N
C
    CALL GEOMTY(L)
C
    DELT = 0.5*DELR*DELR/ALFA
    WRITE(*,120) DELT
    READ(*,*) DELT
    NTIME = TTIME/DELT
    DD = -DELR*DELR/ALFA/DELT
C
C    TIME IS INITIALIZED ... INITIAL CONDITION IS ATTRIBUTED
C
    TIME = 0.0
    DO 10 J=1,N+1
        T(1,J) = BC(0,(J-1)*DELR)
10    CONTINUE
C
    WRITE(20,130) S,S1,S2
    WRITE(*,140)
    READ(*,*) KR
    IF(KR.EQ. 1) THEN

```

```

WRITE(*,150) NTIME
READ(*,*) INTV
ELSEIF(KR .EQ. 2) THEN
    INTV = NTIME
ELSE
    STOP
ENDIF
WRITE(*,160) S,S1,S2
C
C    INITIAL VALUES ARE PRINTED ...
C
    TMEAN = 0.0
    DO 20 J=1,N
        TMEAN = TMEAN + V(J)*(T(1,J) + T(1,J+1))/2.0/V0
20    CONTINUE
    WRITE(*,170) TMEAN,TIME,(T(1,J),J=1,N+1)
    WRITE(20,170) TMEAN,TIME,(T(1,J),J=1,N+1)
C
    IK = 0
C
C    T(1,1) = BC(2,TC)
C    T(1,N+1) = BC(2,T(1,N))
C
C    TIME IS INCREMENTED BY DELT AND BOUNDARY VALUES ARE GIVEN ...
C
30    TIME = TIME + DELT
    IK = IK + 1
C
    CALL COEFF
    CALL TRIDGL(N)
C
    TS(1) = BC(2,TC)
    TS(N+1) = BC(2,TS(N))
C
    TMEAN = 0.0
    DO 40 J=1,N
        TMEAN = TMEAN + V(J)*(TS(J) + TS(J+1))/2.0/V0
40    CONTINUE
C
C    OLD VALUES ARE GIVEN NEW VALUES FOR THE NEXT TIMESTEP ...
C
    DO 50 J=1,N+1
        T(1,J) = TS(J)
50    CONTINUE

```

C

```

IK1 = (IK/INTV)*INTV
IF(IK1 .EQ. IK) THEN
    WRITE(*,170) TMEAN,TIME,(T(1,J),J=1,N+1)
    WRITE(20,170) TMEAN,TIME,(T(1,J),J=1,N+1)
    IF(TIME .LT. TTIME) THEN
        PAUSE ' '
    ELSE
        WRITE(*,*)
        PAUSE 'Going to the MAIN MENU ...'
        GOTO 60
    ENDIF
ENDIF
GOTO 30

```

```

60  CONTINUE

```

C

C

```

***** FORMAT STATEMENTS *****

```

C

```

70  FORMAT(25(/)32X,A15/32X,15('-')/20X,39('-')/21X,'Press [Y]',
1      ' to CONTINUE and [N] to QUIT'/20X,39('-')/)
80  FORMAT(A1)
90  FORMAT(25(/)1X,79('#')/1X,'#',77X,'#/1X,'#',18X,'Solution ',
1      'of Partial Differential Equation',18X,'#/1X,'#',18X,
1      41('-'),18X,'#/1X,'#',77X,'#/1X,'#',31X,A15,31X,'#/
1      1X,'#',77X,'#/1X,'#',24X,'DECLARE PRODUCT CONFIGURA',
1      'TION',24X,'#/1X,'#',24X,29('-'),24X,'#/1X,'#',77X,
1      '#/1X,'#',31X,'1: RECTANGULAR',32X,'#/1X,'#',31X,
1      '2: CYLINDRICAL',32X,'#/1X,'#',31X,'3: SPHERICAL',34X,
1      '#/1X,'#',77X,'#/1X,'#',13X,'[Type any OTHER number',
1      ' to RETURN to the BEGINNING]',14X,'#/1X,'#',77X,'#/
1      1X,79('#')/)
100 FORMAT(25(/)3X,74('-')/13X,A53/13X,53('-')/15X,'Give INITIAL',
1      ' and SIDE 1 temperatures and <ENTER>.'/3X,74('-')/)
110 FORMAT(25(/)1X,78('-')/2X,'Give THICKNESS/RADIUS in [m], ',
1      'TOTAL time [s], No. of DIVISIONS and <ENTER>.'/11X,
1      78('-')/)
120 FORMAT(25(/)18X,41('-')/23X,'Give TIME step in [s] and <ENTER>.'
1      '/19X,'[Lower limit of TIME step = ',F10.5,']/18X,41('-')
1      //)
130 FORMAT(/21X,A15,' - ',A20/21X,38('-')/12X,['',A53,'']/)
140 FORMAT(25(/)19X,42('-')/20X,'Type [1] : to PRINT step-by-step',
1      ' RESULTS'/20X,'Type [2] : to PRINT final RESULTS only.'
1      '/20X,'Type [3] : to STOP'/19X,42('-')/)
150 FORMAT(25(/)5X,71('-')/6X,'Give PRINTING INTERVAL [No. of ',

```

```

1          'TIME steps to be skipped] and <ENTER>.'/25X,['It ',
1          'SHOULD be LESS than ',I5,']'/5X,71('-')/I)
160  FORMAT(20(/)21X,A15,' - ',A20/21X,38('-')/12X,['A53;']/I)
170  FORMAT(/12X,'Mean Temperature = ',F11.5,' at Time = ',F11.5,
1          ' s'/12X,54('-')/6(2X,F11.5))
      STOP
      END
C
C
C  SUBROUTINE TO CALCULATE AREA AND VOLUME OF DIFFERENT SHAPES
C
C
$INCLUDE:'GEOMTY.FOR'
C
C  SUBROUTINE "COEFF" TO SUPPLY VALUES OF COEFFICIENTS TO "TRIDGL" ...
C
C
      SUBROUTINE COEFF
      COMMON /BLOK1/ TIME,TTIME,DELT,DD,TC,T0
      COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
1          TS(101),AR(101),V0,X1,DELR,N
      DOUBLE PRECISION TIME,TTIME,DELR,DELT,DD,TC,T0,C1,C2,C3,CC0
      DOUBLE PRECISION A,B,D,R,V,T,TS,AR,V0,X1,DELR,CC,DDD
C
C  ELEMENTS OF TRIDIAGONAL MATRIX AND CONSTANT VECTOR ARE DEFINED ...
C
      DO 10 J=2,N
          C1 = (AR(J) + AR(J-1))/2.0/AR(J)
          C3 = (AR(J+1) + AR(J))/2.0/AR(J)
          C2 = 2.0*DD - C1 - C3
          DDD = 2.0*DD + C1 + C3
          B(J) = C1
          D(J) = C2
          IF(J .EQ. N) D(J) = C2 + C3
          A(J) = C3
          R(J) = DDD*T(1,J) - C1*T(1,J-1) - C3*T(1,J+1)
10      CONTINUE
C
C  THE FIRST AND THE LAST ELEMENTS OF THE CONSTANT VECTOR ARE
C  CORRECTED ACCORDING TO DIFFERENT BOUNDARY CONDITIONS ...
C
      R(2) = R(2) - B(2)*TC
C
      RETURN

```

```

END
C
C
C   SUBROUTINE "TRIDGL" TO SOLVE A 1-D PDE USING TRIDIGONAL MATRIX ...
C
C
C$INCLUDE:"TRIDGL.FOR"
C
C
C   THIS FUNCTION DEFINES INITIAL AND BOUNDARY CONDITIONS ....
C   (I = 0 FOR INITIAL CONDITION)
C
C
C
C   DOUBLE PRECISION FUNCTION BC(I,TX)
C   COMMON /BLOK1/ TIME,TTIME,DELT,DD,TC1,T0
C   COMMON /PROP/ RO,ALFA,CP,XK,H,PI
C   DOUBLE PRECISION TIME,TTIME,DELT,DD,DD1,DD2,TC1,TC2,T0,TX
C   DOUBLE PRECISION RO,ALFA,CP,XK,H,PI
C
C$INCLUDE:"FCTN.FOR"
C

```

A.3.7.6. Program listing for the solution of partial differential equation by (1) Explicit, (2) Fully Implicit and (3) Crank-Nicolson methods
Program name : **PDEQN.FOR**

```

C
C   THIS PROGRAM COUPLED WITH THE SUBROUTINES 'BOUND','COEFF' AND
C   'TRIDGL' CAN SOLVE A PARTIAL DIFFERENTIAL EQUATION WITH
C   DERIVATIVE OR CONVECTIVE BOUNDARY CONDITION BY (I) EXPLICIT,
C   (II) IMPLICIT AND (III) CRANK-NICOLSON SCHEMES.
C
C   Program Developed by: Prof. Prabir K. Chandra
C
C
C
C   COMMON /BLOK1/ TIME,TTIME,DELT,DD,DD1,DD2,TC1,TC2,L,K,KK
C   COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
1      TS(101),AR(101),V0,X1,DELR,N
C   COMMON /PROP/ T0,RO,ALFA,CP,XK,H1,H2,BETA,PI
C   DOUBLE PRECISION TIME,TTIME,DELR,DELT,DD,DD1,DD2,TC1,TC2,BETA
C   DOUBLE PRECISION T0,RO,ALFA,CP,XK,H1,H2,V0,X1,TMEAN,XM,PI
C   DOUBLE PRECISION A,B,D,R,V,T,TS,AR
C   DOUBLE PRECISION BC

```

CHARACTER S *15, S2 *52

CHARACTER S1 *20

C

OPEN(20,FILE='DATA.OUT',STATUS='NEW')

C

C

KK: INDEX NUMBER [1] EXPLICIT AND [2] IMPLICIT SCHEME AND
[3] CRANK NICHOLSON-METHOD

C

C

K : INDEX NUMBER FOR BOUNDARY CONDITIONS [1] SIDE 1 AT TC1
AND SIDE 2 INSULATED; [2] SIDE 1 AT TC1 AND SIDE 2 AT TC2;

C

C

[3] SIDE 1 AT TC1 AND SIDE 2 CONVECTIVE B.C.; [4] SIDES 1

C

AND 2 CONVECTIVE B.C.; [5] CYLINDRICAL OR SPHERICAL

C

GEOMETRY WITH CONVECTIVE B.C.

C

L : INDEX NUMBER FOR PRODUCT GEOMETRY - [1] RECTANGULAR

C

[2] CYLINDRICAL AND [3] SPHERICAL

C

DELX, DELT, TIME, TTIME : STEP SIZE IN SPACE (m) AND TIME (s),

C

VARIABLE AND TOTAL TIME OF EXPOSURE (s), RESPECTIVELY.

C

D, A, B and N: DIAGONAL, UPPER, LOWER ELEMENTS and ROW OR

C

COLUMN SIZE OF TRIANGULAR MATRIX.

C

R, T: ELEMENTS OF CONSTANT VECTOR AND SOLUTION VECTOR.

C

RO, ALFA, CP, XK, H1, H2 : DENSITY (kg/m^3), DIFFUSIVITY

C

(m^2/s), SPECIFIC HEAT ($\text{J/kg}\cdot\text{C}$), CONDUCTIVITY ($\text{W/m}\cdot\text{C}$)

C

AND HEAT TRANSFER COEFFICIENTS AT SIDE 1 AND 2 ($\text{W/m}^2\cdot\text{C}$).

C

AR(J), V(J) : J-TH AREA PERPENDICULAR TO FLOW (m^2) AND

C

SEGMENT VOLUME (m^3), RESPECTIVELY.

C

TC1,TC2,BETA : TEMPERATURE AT SIDE 1, SIDE 2 (C) AND TEMPERATURE

C

GRADIENT (C/m) [BETA IS ZERO FOR PERFECT INSULATION].

C

DEFINE THERMO-PHYSICAL PROPERTIES ...

C

\$INCLUDE:'THERMO.FOR'

C

PI = 3.141592654

ALFA = XK/RO/CP

H1 = 1.0E-06

H2 = 1.0E-06

TC2 = 1.0E-06

C

DO 60 II=1,100

BETA = 0.0

WRITE(*,70)

READ(*,*) KK

IF(KK.EQ. 1) THEN

S = 'EXPLICIT SCHEME'

ELSEIF(KK.EQ. 2) THEN

```

      S = 'IMPLICIT SCHEME'
ELSEIF(KK.EQ. 3) THEN
      S = 'CRANK-NICOLSON'
ELSE
      CLOSE(20,STATUS='KEEP')
      STOP
ENDIF
WRITE(*,80)
READ(*,*) L
IF(L.EQ. 1) THEN
      S1 = 'RECTANGULAR GEOMETRY'
ELSEIF(L.EQ. 2) THEN
      S1 = 'CYLINDRICAL GEOMETRY'
ELSEIF(L.EQ. 3) THEN
      S1 = 'SPHERICAL GEOMETRY'
ELSE
      GOTO 60
ENDIF
IF(L.EQ. 1) THEN
      WRITE(*,90)
      READ(*,*) K
      IF(K.EQ. 1) THEN
          S2='SIDE 1 : CONSTANT TEMPERATURE; SIDE2 : DERIVATIVE BC'
          WRITE(*,100) S2
          READ(*,*) T0,TC1,BETA
      ELSEIF(K.EQ. 2) THEN
          S2='SIDE 1:CONST. TEMPERATURE; SIDE 2:CONST. TEMPERATURE'
          WRITE(*,110) S2
          READ(*,*) T0,TC1,TC2
      ELSEIF(K.EQ. 3) THEN
          S2='SIDE 1 :CONSTANT TEMPERATURE; SIDE 2:FREE CONVECTION'
          WRITE(*,120) S2
          READ(*,*) T0,TC1,TC2,H2
      ELSEIF(K.EQ. 4) THEN
          S2='SIDE 1 : FREE CONVECTION; SIDE 2 : FREE CONVECTION'
          WRITE(*,130) S2
          READ(*,*) T0,TC1,TC2,H1,H2
      ELSE
          GOTO 60
      ENDIF
ELSEIF(L.GT. 1) THEN
      K = 5
      S2 = 'FREE CONVECTIVE BOUNDARY'
      WRITE(*,140) S2

```



```

        READ(*,*) T0,TC1,H1
        TC2 = TC1
        H2 = H1
    ENDIF
    WRITE(*,150)
    READ(*,*) X1,TTIME,N
    IF(TC1 .EQ. TC2) THEN
        IF((K .EQ. 2) .OR. ((K .EQ. 4) .AND. (H1 .EQ. H2))) THEN
            WRITE(*,160)
            PAUSE ''
            WRITE(20,170)
            DELR = X1/2.0/N
        ELSE
            DELR = X1/N
        ENDIF
    ELSE
        DELR = X1/N
    ENDIF
    IF(L .GE. 2) DELR = X1/N
C
    CALL GEOMTY(L)
C
    DELT = 0.5*DELR*DELR/ALFA
    IF(KK .EQ. 1) THEN
        WRITE(*,180) DELT
    ELSEIF(KK .GE. 2) THEN
        WRITE(*,190) DELT
    ENDIF
    READ(*,*) DELT
    NTIME = TTIME/DELT
    DD = -(DELR*DELR)/ALFA/DELT
    DD1 = XK*(AR(1) + AR(2))/(2.0*AR(1)*H1*DELR)
    DD2 = XK*(AR(N) + AR(N+1))/(2.0*AR(N)*H2*DELR)
C
C    TIME IS INITIALIZED ... INITIAL CONDITION IS ATTRIBUTED
C
    TIME = 0.0
    DO 10 J=1,N+1
        T(1,J) = BC(0,(J-1)*DELR)
        TS(J) = T(1,J)
10    CONTINUE
C
    WRITE(20,200) S,S1,S2
    WRITE(*,210)

```

```

READ(*,*) KR
IF(KR .EQ. 1) THEN
    WRITE(*,220) NTIME,DELT
    READ(*,*) INTV
ELSEIF(KR .EQ. 2) THEN
    INTV = NTIME
ELSE
    STOP
ENDIF
WRITE(*,230) S,S1,S2
TMEAN = 0.0
DO 20 J=1,N
    TMEAN = TMEAN + V(J)*(T(1,J) + T(1,J+1))/2.0/V0
20    CONTINUE

C
C    INITIAL VALUES ARE PRINTED ...
C

    WRITE(*,240) TMEAN,TIME,(T(1,J),J=1,N+1)
    WRITE(20,240) TMEAN,TIME,(T(1,J),J=1,N+1)
    IF(KK .EQ. 1) THEN
        XM = 0.0
    ELSEIF(KK .EQ. 2) THEN
        XM = 1.0
    ELSEIF(KK .EQ. 3) THEN
        XM = 0.5
    ENDIF
    IF(L .EQ. 1) THEN
        IF(K .EQ. 1) THEN
            NN = N + 1
            T(1,N+2) = BC(2,T(1,N)) - 2.0*BETA*DELR
        ENDIF
        IF((K .EQ. 2) .AND. (TC1 .EQ. TC2)) THEN
            NN = N + 1
            T(1,N+2) = BC(2,T(1,N))
        ELSEIF((K.EQ.4).AND.((TC1.EQ.TC2).AND.(H1.EQ.H2))) THEN
            NN = N + 1
            T(1,N+2) = BC(2,T(1,N))
        ELSE
            NN = N
        ENDIF
    ELSE
        NN = N
    ENDIF

C

```

```

                IK = 0
C
                CALL BOUND(NN)
C
C                TIME IS INCREMENTED BY DELT ...
C
30             TIME = TIME + DELT
C
                IK = IK + 1
                CALL COEFF(XM,NN)
                CALL TRIDGL(NN)
C
C                SUBROUTINE BOUND IS CALLED TO ATTRIBUTE BOUNDARY CONDITIONS ...
C
                CALL BOUND(NN)
C
                TMEAN = 0.0
                DO 40 J=1,N
                    TMEAN = TMEAN + V(J)*(TS(J) + TS(J+1))/2.0/V0
40             CONTINUE
C
C                OLD VALUES ARE PUT AS NEW BEFORE GOING FOR THR NEXT TIME STEP ...
C                DO 50 J=1,NN+1
                    T(1,J) = TS(J)
50             CONTINUE
C
                IK1 = (IK/INTV)*INTV
                IF(IK1 .EQ. IK) THEN
                    WRITE(*,240) TMEAN,TIME,(T(1,J),J=1,N+1)
                    WRITE(20,240) TMEAN,TIME,(T(1,J),J=1,N+1)
                    IF(TIME .LT. TTIME) THEN
                        PAUSE ' '
                    ELSE
                        WRITE(*,*)
                        PAUSE ' Going to the MAIN MENU ...'
                        GOTO 60
                   ENDIF
                ENDIF
                GOTO 30
60             CONTINUE
C
C                ***** FORMAT STATEMENTS *****
C
70             FORMAT(15(/)1X,79('#')1X,'#',77X,'#'/1X,'#',18X,'Solution ',

```

```

1      'of Partial Differential Equation',18X,'#/1X,'#/18X,
1      41('-),18X,'#/1X,'#/77X,'#/1X,'#/30X,'MAIN OPTION',
1      'MENU',31X,'#/1X,'#/77X,'#/1X,'#/25X,'CHOOSE ANY ',
1      'OPTION BY NUMBER',25X,'#/1X,'#/25X,27('-),25X,'#/
1      1X,'#/77X,'#/1X,'#/30X,'1: EXPLICIT SCHEME',29X,'#/
1      1X,'#/30X,'2: IMPLICIT SCHEME',29X,'#/1X,'#/30X,
1      '3: CRANK-NICOLSON',30X,'#/1X,'#/77X,'#/1X,'#/23X,
1      '[Type any OTHER number to QUIT]',23X,'#/1X,'#/77X,
1      '#/1X,79('#)//)
80  FORMAT(25(/)1X,79('#)/1X,'#/77X,'#/1X,'#/18X,'Solution ',
1      'of Partial Differential Equation',18X,'#/1X,'#/18X,
1      41('-),18X,'#/1X,'#/77X,'#/1X,'#/31X,'SECONDARY ',
1      'MENU',32X,'#/1X,'#/77X,'#/1X,'#/24X,'DECLARE ',
1      'PRODUCT CONFIGURATION',24X,'#/1X,'#/24X,29('-),
1      24X,'#/1X,'#/77X,'#/1X,'#/31X,'1: RECTANGULAR',
1      32X,'#/1X,'#/31X,'2: CYLINDRICAL',32X,'#/1X,'#/
1      31X,'3: SPHERICAL',34X,'#/1X,'#/77X,'#/1X,'#/12X,
1      '[Type any OTHER number to RETURN to MAIN OPTION MENU]',
1      12X,'#/1X,'#/77X,'#/1X,79('#)//)
90  FORMAT(15(/)1X,79('#)/1X,'#/77X,'#/1X,'#/18X,'Solution ',
1      'of Partial Differential Equation',18X,'#/1X,'#/18X,
1      41('-),18X,'#/1X,'#/77X,'#/1X,'#/31X,'SECONDARY ',
1      'MENU',32X,'#/1X,'#/77X,'#/1X,'#/23X,'CHOOSE ANOT',
1      'HER OPTION BY NUMBER',23X,'#/1X,'#/23X,31('-),23X,
1      '#/1X,'#/77X,'#/1X,'#/10X,'1: SIDE 1-CONSTANT TEM',
1      'PERATURE; SIDE 2-DERIVATIVE BC',15X,'#/1X,'#/10X,'2:',
1      'SIDE 1-CONSTANT TEMPERATURE; SIDE 2-CONSTANT TEMPERAT',
1      'URE',8X,'#/1X,'#/10X,'3: SIDE 1-CONSTANT TEMPERATURE;',
1      ', SIDE 2-FREE CONVECTION',13X,'#/1X,'#/10X,'4: SIDE ',
1      '1-FREE CONVECTION; SIDE 2-FREE CONVECTION',18X,'#/1X,
1      '#/77X,'#/1X,'#/12X,'[Type any OTHER number to RETURN'
1      ', to MAIN OPTION MENU]',12X,'#/1X,'#/77X,'#/1X,
1      79('#)//)
100 FORMAT(25(/)2X,76('-)/14X,A52/14X,52('-)//8X,'Give INITIAL ',
1      'and SIDE 1 temperatures, and GRADIENT and <ENTER>./3X,
1      '[Temperature GRADIENT SHOULD be any constant - ZERO ',
1      'if perfect INSULATION]/2X,76('-)//)
110 FORMAT(25(/)10X,60('-)/14X,A52/14X,52('-)//11X,'Give INITIAL,',
1      'SIDE 1, and SIDE 2 temperatures and <ENTER>./10X,
1      60('-)//)
120 FORMAT(25(/)14X,52('-)/14X,A52/14X,52('-)//16X,'Give INITIAL ',
1      'SIDE 1 and SIDE 2 temperatures and/21X,'HEAT TRANSFER',
1      'COEFFICIENT and <ENTER>./14X,52('-)//)
130 FORMAT(25(/)9X,62('-)/14X,A52/14X,52('-)//10X,'Give INITIAL ',

```

```

1          'temperature, TEMPERATURES at SIDE 1, SIDE 2 and/'
1          11X,'Heat transfer coefficients at SIDE 1 and SIDE 2 and',
1          '<ENTER>.'/9X,62('-')//)
140 FORMAT(25(/)2X,75('-')/14X,A52/14X,52('-')//4X,'Give INITIAL ',
1          'and OUTSIDE temperatures, HEAT TRANSFER COEFF. and ',
1          '<ENTER>.'/2X,75('-')//)
150 FORMAT(25(/)1X,78('-')/2X,'Give THICKNESS/RADIUS in [m], ',
1          'TOTAL time [s], No. of DIVISIONS and <ENTER>.'/11X,
1          78('-')//)
160 FORMAT(25(/)10X,60('-')/12X,'Due to SYMMETRY only HALF of the',
1          ' geometry will be SOLVED.'/10X,60('-')//)
170 FORMAT(/10X,60('-')/12X,'Due to SYMMETRY only HALF of the',
1          ' geometry will be SOLVED.'/10X,60('-')//)
180 FORMAT(25(/)11X,56('-')/23X,'Give TIME step in [s] and <ENTER>'
1          '/12X,'[TIME step should be LESS THAN or EQUAL TO ',F11.5,
1          ']/11X,56('-')//)
190 FORMAT(25(/)18X,41('-')/23X,'Give TIME step in [s] and <ENTER>.'
1          '/19X,'[Lower limit of TIME step = ',F10.5,']/18X,41('-')
1          //)
200 FORMAT(/21X,A15,' - ',A20/21X,38('-')/13X,'[',A52,']/')
210 FORMAT(25(/)19X,42('-')/20X,'Type [1] : to PRINT step-by-step ',
1          'RESULTS'/20X,'Type [2] : to PRINT final RESULTS only.'/
1          20X,'Type [3] : to STOP'/19X,42('-')//)
220 FORMAT(25(/)5X,71('-')/6X,'Give PRINTING INTERVAL [No. of ',
1          'TIME steps to be skipped] and <ENTER>.'/25X,'[It ',
1          'should be LESS than ',I5,']/28X,'[Time step = ',F10.5,
1          ']/5X,71('-')//)
230 FORMAT(20(/)21X,A15,' - ',A20/21X,38('-')/13X,'[',A52,']/')
240 FORMAT(/12X,'Mean Temperature = ',F11.5,' at Time = ',F11.5,
1          ' s'/12X,54('-')//6(2X,F11.5))
      STOP
      END

```

C

C

C THIS SUBROUTINE CALCULATES AREA AND VOLUME FOR DIFFERENT SHAPES

C

C

\$INCLUDE:'GEOMTY.FOR'

C

C

C THIS SUBROUTINE WILL SPECIFY THE BOUNDARY CONDITIONS.

C

C

SUBROUTINE BOUND(NN)

```

COMMON /BLOK1/ TIME, TTIME, DELT, DD, DD1, DD2, TC1, TC2, L, K, KK
COMMON /BLOK2/ A(101), B(101), D(101), R(101), V(101), T(1, 101),
1      TS(101), AR(101), V0, X1, DELR, N
COMMON /PROP/ T0, RO, ALFA, CP, XK, H1, H2, BETA, PI
DOUBLE PRECISION TIME, TTIME, DELR, DELT, DD, DD1, DD2, TC1, TC2, BETA, PI
DOUBLE PRECISION T0, RO, ALFA, CP, XK, H1, H2, V0, X1
DOUBLE PRECISION A, B, D, R, V, T, TS, AR
DOUBLE PRECISION BC

```

C

```

IF(K .EQ. 1) THEN
  TS(1) = BC(2, TC1)
  IF(NN .EQ. N) THEN
    TS(NN+1) = BC(2, TS(NN)) - BETA*DELR
  ELSEIF(NN .EQ. N+1) THEN
    TS(NN+1) = BC(2, TS(N)) - 2.0*BETA*DELR
  ENDIF
ELSEIF(K .EQ. 2) THEN
  IF(TC1 .NE. TC2) THEN
    TS(1) = BC(2, TC1)
    TS(NN+1) = BC(2, TC2)
  ELSE
    TS(1) = BC(2, TC1)
    TS(NN+1) = BC(2, TS(N))
  ENDIF
ELSEIF(K .EQ. 3) THEN
  TS(1) = BC(2, TC1)
  TS(NN+1) = BC(3, TS(NN))
ELSEIF(K .EQ. 4) THEN
  IF((TC1 .EQ. TC2) .AND. (H1 .EQ. H2)) THEN
    TS(1) = BC(1, TS(2))
    TS(NN+1) = BC(2, TS(N))
  ELSE
    TS(1) = BC(1, TS(2))
    TS(NN+1) = BC(3, TS(NN))
  ENDIF
ELSEIF(K .EQ. 5) THEN
  TS(1) = BC(1, TS(2))
  TS(NN+1) = BC(2, TS(NN))
ENDIF

```

C

```

RETURN
END

```

C

C

```

C   SUBROUTINE "COEFF" SUPPLIES COEFFICIENT VALUES TO "TRIDGL"...
C
C
      SUBROUTINE COEFF(XM,NN)
      COMMON /BLOK1/ TIME,TTIME,DELT,DD,DD1,DD2,TC1,TC2,L,K,KK
      COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
1      TS(101),AR(101),V0,X1,DELR,N
      COMMON /PROP/ T0,RO,ALFA,CP,XK,H1,H2,BETA,PI
      DOUBLE PRECISION TIME,TTIME,DELR,DELT,DD,DD1,DD2,TC1,TC2,BETA,PI
      DOUBLE PRECISION T0,RO,ALFA,CP,XK,H1,H2,C1,C11,C3,C33,C22,CC0,CC,
1      DDD,CC1,CC2,AA,XM,V0,X1
      DOUBLE PRECISION A,B,D,R,V,T,TS,AR
C
C   ELEMENTS OF TRIDIAGONAL MATRIX AND CONSTANT VECTOR ARE DEFINED ...
C
      DO 10 J=2,NN
          C1 = (AR(J) + AR(J-1))/2.0/AR(J)
          C11 = XM*C1
          C3 = (AR(J+1) + AR(J))/2.0/AR(J)
          C33 = XM*C3
          C22 = DD - XM*(C1 + C3)
          DDD = DD + (1.0 - XM)*(C1 + C3)
          IF(J .EQ. 2) THEN
              CC0 = C11*DD1/(1.0 + DD1)
              CC = C11*TC1/(1.0 + DD1)
          ENDIF
          IF(J .EQ. NN) THEN
              CC1 = C33*TC2/(1.0 + DD2)
              CC2 = C33*DD2/(1.0 + DD2)
          ENDIF
          B(J) = C11
          D(J) = C22
          IF(J .EQ. NN) THEN
              IF((K.EQ.1).OR.((K.EQ.2).AND.(TC1.EQ.TC2))) THEN
                  IF(NN .EQ. N) THEN
                      D(J) = C22 + C33
                  ELSEIF(NN .EQ. N+1) THEN
                      B(J) = C11 + C33
                  ENDIF
              ENDIF
          ENDIF
          IF(K .EQ. 3) D(J)=C22+CC2
      ENDIF
      IF(K .EQ. 4) THEN
          IF((TC1 .EQ. TC2) .AND. (H1 .EQ. H2)) THEN

```

```

      IF(J .EQ. 2) D(J) = C22 + CC0
      IF(J .EQ. NN) THEN
        IF(NN .EQ. N) THEN
          D(J) = C22 + C33
        ELSEIF(NN .EQ. N+1) THEN
          B(J) = C11 + C33
        ENDIF
      ENDIF
    ELSE
      IF(J .EQ. 2) D(J) = C22 + CC0
      IF(J .EQ. NN) D(J) = C22 + CC2
    ENDIF
  ENDIF
  IF(K .EQ. 5) THEN
    IF(J .EQ. 2) D(J) = C22 + CC0
    IF(J .EQ. NN) D(J) = C22 + C33
  ENDIF
  A(J) = C33
  R(J) = DDD*T(1,J) - (1.0-XM)*(C1*TS(J-1) + C3*TS(J+1))
10  CONTINUE
C
C      THE FIRST AND THE LAST ELEMENTS OF THE CONSTANT VECTOR ARE
C      CORRECTED ACCORDING TO DIFFERENT BOUNDARY CONDITIONS ...
C
      IF(K .EQ. 1) THEN
        R(2) = R(2) - B(2)*TC1
        IF(NN .EQ. N) THEN
          R(NN) = R(NN) + A(NN)*BETA*DELR
        ELSEIF(NN .EQ. N+1) THEN
          R(NN) = R(NN) + 2.0*A(NN)*BETA*DELR
        ENDIF
      ELSEIF(K .EQ. 2) THEN
        IF(TC1 .NE. TC2) THEN
          R(2) = R(2) - B(2)*TC1
          R(NN) = R(NN) - A(NN)*TC2
        ELSE
          R(2) = R(2) - B(2)*TC1
          R(NN) = R(NN) + A(NN)*BETA*DELR
        ENDIF
      ELSEIF(K .EQ. 3) THEN
        R(2) = R(2) - B(2)*TC1
        R(NN) = R(NN) - CC1
      ELSEIF(K .EQ. 4) THEN
        IF((TC1 .EQ. TC2) .AND. (H1 .EQ. H2)) THEN

```



```

      R(2) = R(2) - CC
    ELSE
      R(2) = R(2) - CC
      R(NN) = R(NN) - CC1
    ENDIF
  ELSEIF(K .EQ. 5) THEN
    R(2) = R(2) - CC
  ENDIF
C
  RETURN
END

C
C
C  SUBROUTINE "TRIDGL" TO SOLVE A 1-D PARABOLIC PDE BY TRIDIAGONAL MATRIX.
C
C
C$INCLUDE:'TRIDGL.FOR'
C
C
C  THIS FUNCTION DEFINES INITIAL AND BOUNDARY CONDITIONS ...
C      (I = 0 FOR INITIAL CONDITION)
C
C
C      DOUBLE PRECISION FUNCTION BC(I,TX)
C      COMMON /BLOK1/ TIME,TTIME,DELT,DD,DD1,DD2,TC1,TC2,L,K,KK
C      COMMON /PROP/ T0,RO,ALFA,CP,XK,H1,H2,BETA,PI
C      DOUBLE PRECISION TIME,TTIME,DELT,DD,DD1,DD2,TC1,TC2,T0,RO,
C      1          ALFA,CP,XK,H1,H2,BETA,PI,TX
C
C
C$INCLUDE:'FCTN.FOR'
C

```

A.3.7.7. Program listing for the solution of PDE in two dimensions by Alternating-Direction Implicit (ADI) method.

Program name : **ALTDIR.FOR**

```

C
C  THIS PROGRAM COUPLED WITH THE SUBROUTINES 'BOUND', 'COEFF' AND
C  'TRIDGL' CAN SOLVE A 2-D PARTIAL DIFFERENTIAL EQUATION CONDITION
C  BY THE ALTERNATING-DIRECTION IMPLICIT (ADI) METHOD.
C
C
C      Program Developed by: Prof. Prabir K. Chandra
C

```

```

C
COMMON /BLOK1/ TIME,TTIME,DELX,DELT,D1,D2,DD1,DD2,TC1,TC2,T0,
1      CC1,CC2
COMMON /BLOK2/ A(21),B(21),D(21),R(21),T(21,21,2),TS(21),N
COMMON /PROP/ RO,ALFA,CP,XK,H
DOUBLE PRECISION TIME,TTIME,DELX,DELT,D1,D2,TC1,TC2,T0,DD1,DD2
DOUBLE PRECISION A,B,D,R,T,TS,RO,ALFA,CP,XK,H,X1,TMEAN,DD,CC1,CC2
DOUBLE PRECISION BC
CHARACTER S *37, S2 *56
CHARACTER S1 *22, S0 *1

C
OPEN(20,FILE='DATA.OUT',STATUS='NEW')

C
C      DELX, DELT, TIME,TTIME : STEP SIZE IN SPACE (m) AND TIME (s),
C      VARIABLE AND TOTAL TIME OF EXPOSURE (s), RESPECTIVELY.
C      D, A, B and N: DIAGONAL, UPPER, LOWER ELEMENTS and ROW OR
C      COLUMN SIZE OF TRIANGULAR MATRIX.
C      R, T, TS: ELEMENTS OF CONSTANT VECTOR AND SOLUTION VECTORS.
C      RO, ALFA, CP, XK, H : DENSITY (kg/m^3), DIFFUSIVITY (m^2/s),
C      SPECIFIC HEAT (J/kg.C), CONDUCTIVITY (W/m.C) AND HEAT
C      TRANSFER COEFFICIENT (W/m^2.C).
C      TC1,TC2,T0 : TEMPERATURE AT SIDE 1 AND SIDE 2 AND INITIAL (C).
C
C      DEFINE THERMO-PHYSICAL PROPERTIES ...
C
$INCLUDE:'THERMO.FOR'
C
ALFA = XK/RO/CP
S = 'ALTERNATING-DIRECTION IMPLICIT METHOD'
S1 = '-D PARABOLIC EQUATION'
DO 140 II=1,100
    WRITE(*,150) S
    READ(*,160) S0
    IF((S0.EQ. 'N') .OR. (S0.EQ. 'n')) THEN
        CLOSE(20,STATUS='KEEP')
        STOP
    ENDIF
    S2='X=0 and Y=0:DERIVATIVE; X=L:CONVECTIVE; Y=L:CONST. TEMP.'
    WRITE(*,170) S2
    READ(*,*) T0,TC1,TC2,H
    WRITE(*,180)
    READ(*,*) X1,TTIME,N
    DELX = X1/N      DELT = 0.25*DELX*DELX/ALFA
    WRITE(*,190) DELT

```

```

      READ(*,*) DELT
      NTIME = TTIME/DELT
      DD = DELX*DELX/ALFA/DELT
      D1 = 2.0*(DD + 1.0)
      D2 = 2.0*(DD - 1.0)
      DD1 = XK/H/DELX
      CC1 = TC1*H*DELX/(XK + H*DELX)
      CC2 = XK/(XK + H*DELX)

C
C      TIME IS INITIALIZED ... INITIAL CONDITION IS ATTRIBUTED
C

      TIME = 0.0
      DO 20 I=1,N+1
        DO 10 J=1,N+1
          T(I,J,1) = T0
          T(I,J,2) = T0
10        CONTINUE
20      CONTINUE

C

      WRITE(20,200) S,S1,S2
      WRITE(*,210)
      READ(*,*) KR
      IF(KR .EQ. 1) THEN
        WRITE(*,220) NTIME
        READ(*,*) INTV
      ELSEIF(KR .EQ. 2) THEN
        INTV = NTIME
      ELSE
        STOP
      ENDIF
      WRITE(*,230) S,S1,S2

C
C      INITIAL VALUES ARE PRINTED ...
C

      TMEAN = 0.0
      DO 40 I=1,N
        DO 30 J=1,N
          TMEAN = TMEAN + (T(I,J,1) + T(I,J+1,1))/2.0
30        CONTINUE
40      CONTINUE
      WRITE(*,240) TMEAN/N/N,TIME
      WRITE(20,240) TMEAN/N/N,TIME
      DO 50 I=1,N+1
        WRITE(*,250) (T(I,J,1),J=1,N+1)

```

```

WRITE(20,250) (T(I,J,1),J=1,N+1)
50    CONTINUE
C
    IK = 0
C
    CALL BOUND(2)
C
C    TIME IS INCREMENTED BY DELT AND BOUNDARY VALUES ARE GIVEN ...
C
60    TIME = TIME + DELT
    IK = IK + 1
C
    DO 80 J=1,N
        CALL COEFF(1,J)
        CALL TRIDGL
        CALL BOUND(1)
        DO 70 I=1,N
            T(I,J,1) = TS(I)
70        CONTINUE
80    CONTINUE
C
    DO 100 I=1,N
        CALL COEFF(2,I)
        CALL TRIDGL
        CALL BOUND(2)
        DO 90 J=1,N
            T(I,J,2) = TS(J)
90        CONTINUE
100    CONTINUE
C
    TMEAN = 0.0
    DO 120 I=1,N
        DO 110 J=1,N
            TMEAN = TMEAN + (T(I,J,2) + T(I,J+1,2))/2.0
110        CONTINUE
120    CONTINUE
C
    IK1 = (IK/INTV)*INTV
    IF(IK1 .EQ. IK) THEN
        WRITE(*,240) TMEAN/N/N,TIME
        WRITE(20,240) TMEAN/N/N,TIME
        DO 130 I=1,N+1
            WRITE(*,250) (T(I,J,2),J=1,N+1)
            WRITE(20,250) (T(I,J,2),J=1,N+1)

```

```

130      CONTINUE
      IF (TIME .LT. TTIME) THEN
          PAUSE ' '
      ELSE
          WRITE(*,*)
          PAUSE 'Going to the MAIN MENU ...'
          GOTO 140
      ENDIF
    ENDIF
    GOTO 60
140 CONTINUE
C
C ***** FORMAT STATEMENTS *****
C
150 FORMAT(25(/)21X,A37/20X,39('-')//21X,'Press [Y] to CONTINUE',
1      ' and [N] to QUIT'/20X,39('-')//)
160 FORMAT(A1)
170 FORMAT(25(/)7X,65('-')/12X,A56/12X,56('-')/8X,'Give INITIAL',
1      ' temperature, temperature at SIDES X=L and Y=L, and'/
1      21X,'HEAT transfer coefficient and <ENTER>.'/7X,65('-')
1      //)
180 FORMAT(25(/)2X,76('-')/3X,'Give LENGTH or WIDTH in [m], ',
1      ' TOTAL time [s], No. of DIVISIONS and <ENTER>.'/2X,
1      76('-')//)
190 FORMAT(25(/)18X,41('-')/23X,'Give TIME step in [s] and <ENTER>.'
1      '/19X,['Lower limit of TIME step = ',F10.5,']/18X,41('-')
1      //)
200 FORMAT(/9X,A37,' - ',A22/9X,62('-')/11X,[' ',A56,'/'])
210 FORMAT(25(/)19X,42('-')/20X,'Type [1] : to PRINT step-by-step',
1      ' RESULTS'/20X,'Type [2] : to PRINT final RESULTS only.'
1      '/20X,'Type [3] : to STOP'/19X,42('-')//)
220 FORMAT(25(/)5X,71('-')/6X,'Give PRINTING INTERVAL [No. of ',
1      ' TIME steps to be skipped] and <ENTER>.'/25X,['It ',
1      ' SHOULD be LESS than ',I5,']/5X,71('-')//)
230 FORMAT(20(/)9X,A37,' - ',A22/9X,62('-')/11X,[' ',A56,'/'])
240 FORMAT(/12X,'Mean Temperature = ',F11.5,' at Time = ',F11.5,
1      ' s'/12X,54('-')//)
250 FORMAT(6(2X,F11.5))
      STOP
      END

```

C

C

C THIS SUBROUTINE SUPPLIES BOUNDARY CONDITIONS TO THE MAIN PROGRAM

C

C

```

SUBROUTINE BOUND(M)
COMMON /BLOK1/ TIME,TTIME,DELX,DELT,D1,D2,DD1,DD2,TC1,TC2,T0,
1          CC1,CC2
COMMON /BLOK2/ A(21),B(21),D(21),R(21),T(21,21,2),TS(21),N
COMMON /PROP/ RO,ALFA,CP,XK,H
DOUBLE PRECISION TIME,TTIME,DELX,DELT,D1,D2,TC1,TC2,T0,DD1,DD2
DOUBLE PRECISION A,B,D,R,T,TS,RO,ALFA,CP,XK,H,CC1,CC2
DOUBLE PRECISION BC

```

C

```

DO 10 I=1,N+1
      T(N+1,I,M) = BC(1,T(N,I,M))
      T(I,N+1,M) = BC(2,TC2)

```

10 CONTINUE

C

```

RETURN
END

```

C

C

THIS SUBROUTINE CALCULATES ALL THE COEFFICIENTS FOR "TRIDGL"

C

C

```

SUBROUTINE COEFF(M,L)
COMMON /BLOK1/ TIME,TTIME,DELX,DELT,D1,D2,DD1,DD2,TC1,TC2,T0,
1          CC1,CC2
COMMON /BLOK2/ A(21),B(21),D(21),R(21),T(21,21,2),TS(21),N
DOUBLE PRECISION TIME,TTIME,DELX,DELT,D1,D2,TC1,TC2,T0,AA,DD1,
1          DD2,CC1,CC2
DOUBLE PRECISION A,B,D,R,T,TS

```

C

C ELEMENTS OF TRIDIAGONAL MATRIX AND CONSTANT VECTOR ARE DEFINED ...

C

```

DO 10 I=1,N
      A(I) = -1.0
      B(I) = -1.0
      D(I) = D1

```

10 CONTINUE

```

A(1) = -2.0
IF(L.EQ. 1) THEN
      D(N) = D(N) - CC2
ENDIF

```

C

C

THE FIRST AND THE LAST ELEMENTS OF THE CONSTANT VECTOR ARE

C CORRECTED ACCORDING TO BOUNDARY CONDITIONS ...

C

DO 20 I=1,N

IF(M .EQ. 1) THEN

IF(L .EQ. 1) THEN

$R(I) = D2 * T(I,1,2) + 2.0 * T(I,2,2)$

ELSE

$R(I) = T(I,L-1,2) + D2 * T(I,L,2) + T(I,L+1,2)$

ENDIF

ELSEIF(M .EQ. 2) THEN

IF(L .EQ. 1) THEN

$R(I) = D2 * T(1,I,1) + 2.0 * T(2,I,1)$

ELSE

$R(I) = T(L-1,I,1) + D2 * T(L,I,1) + T(L+1,I,1)$

ENDIF

ENDIF

20 CONTINUE

C

IF(M .EQ. 1) THEN

$R(N) = R(N) + CC1$

ELSEIF(M .EQ. 2) THEN

$R(N) = R(N) + TC2$

ENDIF

C

RETURN

END

C

C

C THIS SUBROUTINE SOLVES A 2-D PARABOLIC PDE USING TRIDIAGONAL MATRIX

C

C

SUBROUTINE TRIDGL

COMMON /BLOK2/ A(21),B(21),D(21),R(21),T(21,21,2),TS(21),N

DOUBLE PRECISION A,B,D,R,T,TS,AA

C

$A(N) = 0.0$

$B(1) = 0.0$

$B(N) = B(N)/D(N)$

$R(N) = R(N)/D(N)$

C

DO 10 I=2,N

$J = N - I + 2$

$AA = 1.0 / (D(J-1) - B(J) * A(J-1))$

$B(J-1) = B(J-1) * AA$

```

        R(J-1) = (R(J-1) - A(J-1)*R(J))*AA
10    CONTINUE
C
C    BACK SUBSTITUTION STARTS ...
C
        TS(1) = R(1)
        DO 20 J=2,N
            TS(J) = R(J) - B(J)*TS(J-1)
20    CONTINUE
C
        RETURN
        END
C
C
C    THIS FUNCTION DEFINES INITIAL AND BOUNDARY CONDITIONS
C    (WHEN I = 0, IT IS INITIAL CONDITION)
C
C
        DOUBLE PRECISION FUNCTION BC(I,TX)
        COMMON /BLOK1/ TIME,TTIME,DELX,DELT,D1,D2,DD1,DD2,TC1,TC2,T0,
I          CC1,CC2
        COMMON /PROP/ RO,ALFA,CP,XK,H
        DOUBLE PRECISION TIME,TTIME,DELX,DELT,D1,D2,TC1,TC2,T0,TX,DD1,DD2
        DOUBLE PRECISION RO,ALFA,CP,XK,H,CC1,CC2
C
$INCLUDE:'FCTN.FOR'
C

```

A.3.7.8. Program listing **THERMO.FOR** to be included in all the programs listed.

Program name : **THERMO.FOR**

```

C
C
C    THERMOPHYSICAL PROPERTIES OF MATERIAL
C
C
        RO = 1050.0
        CP = 3500.0
        XK = 1.5
C

```


A.3.7.9. Program listing **FCTN.FOR** to be included in the programs listed in Appendices A.3.7.1, A.3.7.3 and A.3.7.4.

Program name : **FCTN.FOR**

```
C
C
C  THIS FUNCTION DEFINES INITIAL AND BOUNDARY CONDITIONS ...
C  [WHEN I = 0, IT SPECIFIES INITIAL CONDITION, OTHERWISE BC].
C
C
C      IF(I .EQ. 0) THEN
C          BC = T0
C      ELSEIF(I .EQ. 1) THEN
C          BC = (TC1 + DD1*TX)/(1.0 + DD1)
C      ELSEIF(I .EQ. 2) THEN
C          BC = TX
C      ELSEIF(I .EQ. 3) THEN
C          BC = (TC2 + DD2*TX)/(1.0 + DD2)
C      ENDIF
C
C      RETURN
C      END
C
```

APPENDIX A.5. Program listings for solutions of application problems.

A.5.1. Complete program listing for psychrometric functions.

Program name : **PSYCRO.FOR**

```
C
C
C FUNCTION PST(T) TO COMPUTE SATURATION VAPOR PRESSURE IN Pa
C
C
C DOUBLE PRECISION FUNCTION PST(T)
C DOUBLE PRECISION F(8),T,SUM
C
C F(1) = -741.9242
C F(2) = -29.721
C F(3) = -11.55286
C F(4) = -0.8685635
C F(5) = 0.1094098
C F(6) = 0.439993
C F(7) = 0.2520658
C F(8) = 0.05218684
C
C SUM = 0.0
C DO 10 I=1,8
C SUM = SUM + F(I)*(0.65 - 0.01*T)**(I-1)
10 CONTINUE
C PST = 22087836.75*DEXP((0.01/(T+273.15))*(374.136 - T)*SUM)
C RETURN
C END
C
C
C FUNCTION VTH(T,H) COMPUTES HUMID VOLUME AS A FUNCTION OF
C TEMPERATURE AND ABSOLUTE HUMIDITY IN m^3 OF MOIST PER kg
C DRY AIR
C
C
C DOUBLE PRECISION FUNCTION VTH(T,H)
C COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
C DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,T,H
C
C VTH = R*(T+273.15)*(1.0 + (WMA/WMV)*H)/P
C RETURN
C END
C
C
C FUNCTION ROTH(T,H) TO CALCULATE DENSITY AS A FUNCTION OF
C TEMPERATURE AND ABSOLUTE HUMIDITY, IN kg MOIST/m^3 MOIST AIR
C
C
C DOUBLE PRECISION FUNCTION ROTH(T,H)
```

```

DOUBLE PRECISION VTH,T,H
C
ROTH = (1.0 + H)/VTH(T,H)
RETURN
END
C
C
C FUNCTION PVH(H) COMPUTES VAPOR PRESSURE AS A FUNCTION OF
C ABSOLUTE HUMIDITY, IN Pa
C
C
DOUBLE PRECISION FUNCTION PVH(H)
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,H
C
PVH = P*H/((WMV/WMA) + H)
RETURN
END
C
C
C FUNCTION RHHT(H,T) COMPUTES RELATIVE HUMIDITY AS A
C FUNCTION OF ABSOLUTE HUMIDITY AND TEMPERATURE, FRACTION
C
C
DOUBLE PRECISION FUNCTION RHHT(H,T)
DOUBLE PRECISION PVH,PST,H,T
C
RHHT = PVH(H)/PST(T)
RETURN
END
C
C
C FUNCTION HDBWB(DB,WB) CALCULE ABSOLUT HUMIDITY AS A
C FUNCTION OF DRY AND WET BULB TEMPERATURES, kg VAPOR/kg
C DRY AIR
C
C
DOUBLE PRECISION FUNCTION HDBWB(DB,WB)
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION HPV,PST,P,R,WMA,WMV,HC,CPA,CPV,CPL,X,Y,DB,WB
C
X = (HC - (CPL - CPV)*WB)*HPV(PST(WB)) - CPA*(DB-WB)
Y = HC + CPV*DB - CPL*WB
HDBWB = X/Y
RETURN
END
C
C
C FUNCTION HPV(PW) COMPUTES ABSOLUTE HUMIDITY A FUNCTION
C OF VAPOR PRESSURE OF WATER, kg VAPOR/kg DRY AIR
C
C
DOUBLE PRECISION FUNCTION HPV(PW)

```

```

COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,PW
C
  HPV = (WMV/WMA)*PW/(P - PW)
  RETURN
  END
C
C
C
C FUNCTION VST(T) COMPUTES ABSOLUTE VISCOSITY OF DRY AIR AS A
C FUNCTION OF TEMPERATURE, IN Pa.s
C
C
C
C DOUBLE PRECISION FUNCTION VST(T)
C DOUBLE PRECISION T
C
C
C VST = (1.7471 + 4.0181E-03*T)/1.0D05
C RETURN
C END
C
C
C
C FUNCTION PSHRH(H,UR) COMPUTES VAPOR PRESSURE AS A
C FUNCTION OF ABSOLUTE HUMIDITY AND RELATIVE HUMIDITY, IN Pa
C
C
C
C DOUBLE PRECISION FUNCTION PSHRH(H,UR)
C DOUBLE PRECISION PVH,H,UR
C
C
C PSHRH = PVH(H)/UR
C RETURN
C END
C
C
C
C FUNCTION RHPVT(PW,T) TO COMPUTE RELATIVE HUMIDITY AS A
C FUNCTION OF VAPOR PRESSURE AND TEMPERATURE, FRACTION
C
C
C
C DOUBLE PRECISION FUNCTION RHPVT(PW,T)
C DOUBLE PRECISION PST,PW,T
C
C
C RHPVT = PW/PST(T)
C RETURN
C END
C
C
C
C FUNCTION ENTH(T,H) TO CALCULATE ENTHALPY AS A FUNCTION
C OF TEMPERATURE AND ABSOLUTE HUMIDITY, IN J/kg
C
C
C
C DOUBLE PRECISION FUNCTION ENTH(T,H)
C COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
C DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,T,H
C
C
C ENTH = CPA*T + H*(HC + CPV*T)

```

RETURN
END

C

C

C FUNCTION TDTWH(TU,H) COMPUTES DRY BULB TEMPERATURE AS A
C FUNCTION OF ABSOLUTE HUMIDITY AND WET BULB TEMPERATURE,
C IN C

C

C

DOUBLE PRECISION FUNCTION TDTWH(TU,H)
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION HPV,PST,P,R,WMA,WMV,HC,CPA,CPV,CPL,X,Y,TU,H

C

$X = HC * H - CPL * TU * H - CPA * TU$
 $Y = (HC - (CPL - CPV) * TU) * HPV(PST(TU))$
 $TDTHW = (Y - X) / (CPA + CPV * H)$
RETURN
END

C

C

C FUNCTION TDHV(H,V) COMPUTES DRY BULB TEMPERATURE AS A
C FUNCTION OF ABSOLUTE HUMIDITY AND HUMID VOLUME, IN C

C

C

DOUBLE PRECISION FUNCTION TDHV(H,V)
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,H,V

C

$TDHV = P * V / (R * (1.0 + (WMA / WMV) * H)) - 273.15$
RETURN
END

C

C

C FUNCTION TDHE(H,E) COMPUTES DRY BULB TEMPERATURE AS A
C FUNCTION OF ABSOLUTE HUMIDITY AND ENTHALPY, IN C

C

C

DOUBLE PRECISION FUNCTION TDHE(H,E)
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,H,E

C

$TDHE = (E - HC * H) / (CPA + CPV * H)$
RETURN
END

C

C

C FUNCTION HTV(T,V) TO CALCULATE ABSOLUTE HUMIDITY AS A
C FUNCTION OF TEMPERATURE AND HUMID VOLUME, kg VAPOR/kg
C DRY AIR

C

C

DOUBLE PRECISION FUNCTION HTV(T,V)
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL

```

DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,T,V
C
HTV = (P*V - R*(T+273.15))/((WMA/WMV)*R*(T+273.15))
RETURN
END
C
C
C FUNCTION HTE(T,E) TO COMPUTE ABSOLUTE HUMIDITY AS A
C FUNCTION OF TEMPERATURE AND ENTHALPY, kg VAPOR/kg DRY AIR
C
C
DOUBLE PRECISION FUNCTION HTE(T,E)
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,T,E
C
HTE = (E - CPA*T)/(HC + CPV*T)
RETURN
END
C
C
C FUNCTION HLT(T) COMPUTES LATENT HEAT OF EVAPORATION AS A
C FUNCTION OF TEMPERATURE, IN J/kg
C
C
DOUBLE PRECISION FUNCTION HLT(T)
DOUBLE PRECISION T1,HLT1,T
C
T1 = (1.8*T + 32.0) + 459.69
IF((T1 .GE. 459.69) .AND. (T1 .LT. 491.69)) THEN
    HLT1 = 1220.884 - 0.05077*(T1 - 459.69)
ELSEIF((T1 .GE. 491.69) .AND. (T1 .LT. 609.69)) THEN
    HLT1 = 1075.8965 - 0.56983*(T1 - 459.69)
ELSEIF(T1 .GE. 609.69) THEN
    HLT1 = DSQRT(1354673.214 - 0.9125275587*T1*T1)
ENDIF
HLT = 2326.0*HLT1
RETURN
END
C
C
C FUNCTION HQH(H) COMPUTES HUMID HEAT AS A FUNCTION OF
C ABSOLUTE HUMIDITY, IN J/kg.C
C
C
DOUBLE PRECISION FUNCTION HQH(H)
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,H
C
HQH = CPA + CPV*H
RETURN
END
C

```

A.5.2. Complete program to calculate psychrometric properties if any two are given.

Program name: **PSY2000.FOR**

```

C
C
C THIS MAIN PROGRAM CALCULATES ALL THE PSYCHROMETRIC
C PROPERTIES OF AIR WHEN ANY TWO PROPERTIES ARE GIVEN.
C
C
COMMON /BLK1/ X1,X2,X3,X4,TD,TW,TP,HU,RH,VH,EN,RO,PV,SP,HL,QH,VI
COMMON /BLK2/ P0,P1,P2,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
COMMON /BLK3/ L,KK,M,N,T1,T2,P3
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION X1,X2,X3,X4,TD,TW,TP,HU,RH,VH,EN,RO,PV,SP,HL
DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,T1,T2,P3,QH,VI
DOUBLE PRECISION PST,PSHRH,HPV,HDBWB,PVH,RHPVT,RHHT,VTH,ENTH
DOUBLE PRECISION ROTH,TDTHW,TDHV,TDHE,HTV,HTE,HLT,HQH,VST
CHARACTER *2 P0,P1,P2,Y
CHARACTER *17 A1,A2
CHARACTER *9 A4,A5,A6,A7,A8,A9,A10
CHARACTER *20 A3
EXTERNAL PROP,TDS
C
C ALL THE PROPERTIES OF AIR ARE SPECIFIED ...
C
R = 287.055724249
CPA = 1004.832
WMA = 28.9645
C
WRITE(*,40)
READ(*,50) A3
OPEN(10,FILE=A3,STATUS='NEW')
WRITE(*,60)
WRITE(*,70)
READ(*,80) Y
IF((Y .EQ. 'Y') .OR. (Y .EQ. 'y')) THEN
    WRITE(*,90)
    PAUSE ''
ENDIF
WRITE(*,100)
READ(*,*) KK
A5 = '[Pa]'
IF(KK .EQ. 2) THEN
    A5 = '[mm Hg]'
ELSEIF(KK .EQ. 3) THEN
    A5 = '[psi]'

```

```

ENDIF
WRITE(*,110) A5
READ(*,120) P0,P3
CALL TITLE(1)
DO 20 I=1,100
    L = 1
    M = 0
    T1 = 20.0
    T2 = 50.0
    CALL TITLE(2)
    READ(*,150) P1,P2
    CALL VALUES(L, KK)
    IF(L .LT. 17) THEN
        WRITE(*,160) P1,P2
        READ(*,*) X1,X2
        X3 = X1
        X4 = X2
        CALL VALUES(L, KK)
10    CALL SELECT(L, KK, M, N, T1, T2, PROP, TDS)
        IF(N .EQ. 5000) WRITE(*,170) P1,P2,N
        IF(M .EQ. 5000) THEN
            WRITE(*,180) P1,P2,M,T1,T2
            READ(*,*) T1,T2
            IF((T1 .LT. 0.0) .OR. (T2 .LT. 0.0)) GOTO 20
            M = 0
            N = 0
            GOTO 10
        ENDIF
    ELSE
        WRITE(*,190) A1,A2
        WRITE(10,190) A1,A2
        GOTO 20
    ENDIF
    CALL TITLE(3)
    WRITE(*,130)
    READ(*,140) Y
    IF((Y .EQ. 'N') .OR. (Y .EQ. 'n')) GOTO 30
20 CONTINUE
C
30 CALL TITLE(4)
    CLOSE(10,STATUS='KEEP')
C
C F O R M A T   S T A T E M E N T S
C
40 FORMAT(////////1X,79('#)/1X,'#',77X,'#/1X,'#',28X,'PSYCHRO',
1      'METRIC PACKAGE',28X,'#/1X,'#',28X,21('-'),28X,'#/1X,'#'
2      ',16X,'Program developed by Prof. Prabir K. Chandra',
3      '17X,'#/1X,'#',34X,'DCA/ESAL',35X,'#/1X,'#',77X,'#/1X,
4      '#',21X,'Dedicated to : My Parents and Family',20X,'#/1X,
5      '#',77X,'#/1X,79('#)//5X,'Type a name of the OUTPUT ',
6      'file : [Ex. C:\MODEL\DATA.OUT or A:DATA.OUT]'/23X,
7      'within 20 CHARACTERS and <ENTER>.'////)
50 FORMAT(A20)

```



```

60 FORMAT(20(/),27X,'UNITS IN DIFFERENT SYSTEMS'/27X,
1      26('-')/2X,78('_')/20X,'S.I.',20X,'METRIC',20X,'BRIT',
1      'ISH'/2X,78('-')/2X,'Temperature',8X,'C',24X,'C',26X,'F'
1      /2X,78('-')/2X,'Pressure',11X,'Pa',19X,'mm mercury',20X,
1      'psi'/2X,78('-')/2X,'Enthalpy/L. heat J/kg',20X,'kcal',
1      '/kg',20X,'Btu/lb')
70 FORMAT(2X,78('-')/2X,'Humid heat',7X,'J/kg.C',18X,'kcal/kg.C',
1      19X,'Btu/lb.F'/2X,78('-')/2X,'Density',10X,'kg/m^3',19X,
1      'kg/m^3',21X,'lb/ft^3'/2X,78('-')/2X,'Sp. volu',
1      'me',7X,'m^3/kg',19X,'m^3/kg',21X,'ft^3/lb'/2X,78('-')/
1      2X,'Viscosity',9X,'Pa.s',16X,'centipoise(cP)',17X,
1      'lb/ft.s'/2X,78('_')/2X,'If want to refer to CONVERS',
1      'ION TEABLE, Type [Y]; if NOT [N] and <ENTER>./')
80 FORMAT(A2)
90 FORMAT(20(/),30X,'CONVERSION TABLE'/30X,16('-')/2X,78('_')//
1      2X,'[F = C*1.8 + 32; C = (F-32)/1.8]'/2X,'[1 atm = ',
1      '101325 Pa = 760 mm mercury = 14.69594878 psi = ',
1      '1033.2348 cm water]'/2X,'[1 Btu/lb = 2326.0 J/kg = ',
1      '0.55555556 kcal/kg]'/2X,'[1 Btu/lb.F = 4186.8 J/kg.C',
1      ' = 1 kcal/kg.C]'/2X,'[1 lb/ft^3 = 16.01846337 kg/m^3;',
1      ' 1 kg/m^3 = 0.06242796 lb/ft^3]'/2X,'[1 ft^3/lb = ',
1      '0.06242796 m^3/kg; 1 m^3/kg = 16.01846337 ft^3/lb]'/
1      2X,'[1 lb/ft.s = 1.488163944 Pa.s = 1488.163944 cP]'/
1      2X,78('_'))
100 FORMAT(20(/)12X,55('-')/23X,'Type [1] or [2] or [3] and ',
1      '<ENTER>'/13X,'1 : S.I. SYSTEM; 2 : METRIC SYSTEM; 3',
2      ': BRITISH SYSTEM'/12X,55('-')////)
110 FORMAT(20(/)2X,78('-')/4X,'Now type ONE of the two-letter ',
1      'abbreviations given below and atmospheric'/9X,
1      'PRESSURE in ',A7,' using floating point notation and',
1      '<ENTER>.'/2X,AA : Air-Alcohol; AB : Air-Benzene; ',
1      'AT : Air-Toluene; AW : Air-Water System.'/2X,78('_')
1      ///)
120 FORMAT(A2,1X,F12.5)
130 FORMAT(20(/)15X,50('-')/16X,'Type [Y] to CONTINUE and [N] ',
1      'to QUIT and <ENTER>'/15X,50('-')////)
140 FORMAT(A2)
150 FORMAT(A2,1X,A2)
160 FORMAT(/9X,61('-')/10X,'Give the VALUES of ',A2,' and ',
1      A2,' in specified UNITS and <ENTER>'/9X,61('-')//)
170 FORMAT(20(/)1X,77('-')/2X,'Probably the scheme with ',A2,
1      ' and ',A2,' is NOT converging within ',I4,
2      ' iterations'/1X,77('-')////)
180 FORMAT(20(/)4X,72('-')/5X,'The scheme with Ps or Lh or Mu ',
1      'in case of ',A2,' and ',A2,' is NOT converging.'/8X,
2      'No. of iterations was : ',I4,'. Guesses of T1 and T2 ',
3      'were : ',F6.2/13X,'and ',F6.2,'. Give other ',
4      'guesses of T1 and T2 (T2 > T1)/15X,'or a NEGATIVE ',
5      'value of T1 or T2 to go to MAIN MENU.'/4X,72('_')////)
190 FORMAT(/9X,'This combination of ',A17,'and ',A17/9X,'does not',
1      ' have a SOLUTION.'/)
STOP
END

```

```

C
C
C   THIS SUBROUTINE WRITES VARIOUS TITLES OF THE PROGRAM
C
C
SUBROUTINE TITLE(I)
COMMON /BLK1/ X1,X2,X3,X4,TD,TW,TP,HU,RH,VH,EN,RO,PV,SP,HL,QH,VI
COMMON /BLK2/ P0,P1,P2,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
COMMON /BLK3/ L,KK,M,N,T1,T2,P3
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION X1,X2,X3,X4,TD,TW,TP,HU,RH,VH,EN,RO,PV,SP,HL
DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,T1,T2,P3,QH,VI
CHARACTER *2 P0,P1,P2,Y
CHARACTER *17 A1,A2
CHARACTER *9 A4,A5,A6,A7,A8,A9,A10
CHARACTER *20 A3

C
IF(I .EQ. 1) THEN
  P = P3
  A4 = 'C'
  A5 = 'Pa'
  A6 = 'J/kg'
  A7 = 'J/kg.C'
  A8 = 'kg/m^3'
  A9 = 'm^3/kg'
  A10 = 'Pa.s'
IF(KK .EQ. 2) THEN
  P = 133.3223684*P3
  A4 = 'C'
  A5 = 'mm Hg'
  A6 = 'kcal/kg'
  A7 = 'kcal/kg.C'
  A8 = 'kg/m^3'
  A9 = 'm^3/kg'
  A10 = 'cP'
ELSEIF(KK .EQ. 3) THEN
  P = 6894.757291*P3
  A4 = 'F'
  A5 = 'psi'
  A6 = 'Btu/lb'
  A7 = 'Btu/lb.F'
  A8 = 'lb/ft^3'
  A9 = 'ft^3/lb'
  A10 = 'lb/ft.s'
ENDIF
IF((P0 .EQ. 'AW') .OR. (P0 .EQ. 'aw')) THEN
  WMV = 18.01534
  HC = 2500915.2
  CPV = 1858.9392
  CPL = 4186.8
  A3 = 'AIR - WATER SYSTEM'
ELSEIF((P0 .EQ. 'AB') .OR. (P0 .EQ. 'ab')) THEN
  WMV = 21.5

```

```

      HC = 2500000.0
      CPV = 1800.0
      CPL = 3500.0
      A3 = 'AIR - BENZENE SYSTEM'
ELSE
      WMV = 18.051534
      HC = 2500915.2
      CPV = 1858.9392
      CPL = 4186.8
      A3 = ' AIR - WATER SYSTEM'
ENDIF
WRITE(10,10) P3,A5,A3
ELSEIF(I.EQ. 2) THEN
      WRITE(*,20) A4,A4,A4,A5,A5,A9,A6,A6,A7,A10,A8
ELSEIF(I.EQ. 3) THEN
      IF(KK.EQ. 1) THEN
            WRITE(*,30) TD,TW,TP,HU,PV,SP,RH*100.0,VH
            WRITE(*,40) EN,HL,QH,VI,RO,P1,X3,P2,X4
            WRITE(10,30) TD,TW,TP,HU,PV,SP,RH*100.0,VH
            WRITE(10,40) EN,HL,QH,VI,RO,P1,X3,P2,X4
      ELSEIF(KK.EQ. 2) THEN
            WRITE(*,50) TD,TW,TP,HU,PV,SP,RH*100.0,VH
            WRITE(*,60) EN,HL,QH,VI,RO,P1,X3,P2,X4
            WRITE(10,50) TD,TW,TP,HU,PV,SP,RH*100.0,VH
            WRITE(10,60) EN,HL,QH,VI,RO,P1,X3,P2,X4
      ELSEIF(KK.EQ. 3) THEN
            WRITE(*,70) TD,TW,TP,HU,PV,SP,RH*100.0,VH
            WRITE(*,80) EN,HL,QH,VI,RO,P1,X3,P2,X4
            WRITE(10,70) TD,TW,TP,HU,PV,SP,RH*100.0,VH
            WRITE(10,80) EN,HL,QH,VI,RO,P1,X3,P2,X4
      ENDIF
      PAUSE ''
ELSEIF(I.EQ. 4) THEN
      WRITE(10,90) A4,A4,A4,A5,A5,A9,A6,A6,A7,A10,A8
      WRITE(10,100)
      WRITE(*,110)
ENDIF
C
C      ***** FORMAT STATEMENTS *****
C
10 FORMAT(/23X,'Atmospheric pressure = ',F8.1,1X,A9///17X,'CALCUL'
1      'ATED PROPERTIES FOR ',A20/17X,46('-')/)
20 FORMAT(/////2X,78('_')/2X,78('-')/32X,'MAIN MENU'/32X,10('-')
1      //4X,'Type any TWO 2-letter abbreviations listed below '
1      ',with a space or a comma'/13X,'in-between and press ',
1      '<ENTER>. (Use ONLY CAPITAL letters)'/15X,
1      'TD : Dry bulb temperature, ',A9/ 15X,'TW : Wet bulb ',
1      'Temperature, ',A9/15X,'TP : Dew-point temperature, ',A9
1      '/15X,'HU : Absolute humidity, fraction'/15X,'PV : Vapor'
1      ', pressure at TD, ',A9/15X,'PS : Saturation vapor pre',
1      'ssure at TD, ',A9/15X,'RH : Relative humidity, %'/15X,
1      'VH : Humid volume or Specific volume, ',A9/15X,'ÉN :',
1      'Enthalpy, ',A9/15X,'LH : Latent heat of vaporization,')

```

```

1          ,IX,A9/15X,'QH : Humid heat, ',A9/15X,'MU : Viscosity,',
1          1X,A9/15X,'RO : Density, ',A9/2X,78(' ')/)
30 FORMAT(1X,79(' ')/6X,'Td',6X,'Tw',6X,'Tp',7X,'Hu',8X,'Pv',9X,
1          'Ps',9X,'Rh',7X,'Vh'/1X,79(' ')/3X,F6.2,2X,F6.2,2X,F6.2,
2          2X,F8.6,3X,F8.2,2X,F9.2,2X,F8.4,2X,F8.6)
40 FORMAT(1X,79(' ')/15X,'En',10X,'Lh',11X,'Qh',10X,'Mu',9X,'Ro'/
1          10X,60(' ')/10X,F10.2,3X,F10.2,2X,F10.4,1X,F12.10,2X,
2          F8.6/10X,60(' ')/13X,'Properties given : ',A2,' =',
3          1PD13.6,' ',A2,' =',1PD13.6,' ')/1X,79(' ')/1X,79(' ')/)
50 FORMAT(1X,79(' ')/6X,'Td',6X,'Tw',6X,'Tp',7X,'Hu',8X,'Pv',9X,
1          'Ps',8X,'Rh',8X,'Vh'/1X,79(' ')/3X,F6.2,2X,F6.2,2X,F6.2,
2          2X,F8.6,1X,F10.5,1X,F10.5,2X,F8.4,2X,F8.6)
60 FORMAT(1X,79(' ')/15X,'En',10X,'Lh',11X,'Qh',10X,'Mu',9X,'Ro'/
1          10X,60(' ')/10X,F10.6,2X,F11.6,1X,F10.6,2X,F12.10,2X,
2          F8.6/10X,60(' ')/13X,'Properties given : ',A2,' =',
3          1PD13.6,' ',A2,' =',1PD13.6,' ')/1X,79(' ')/1X,79(' ')/)
70 FORMAT(1X,79(' ')/6X,'Td',6X,'Tw',6X,'Tp',7X,'Hu',8X,'Pv',9X,
1          'Ps',9X,'Rh',7X,'Vh'/1X,79(' ')/3X,F6.2,2X,F6.2,2X,F6.2,
2          2X,F8.6,2X,F8.6,2X,F9.6,4X,F8.4,1X,F8.4)
80 FORMAT(1X,79(' ')/15X,'En',10X,'Lh',11X,'Qh',10X,'Mu',9X,'Ro'/
1          10X,60(' ')/10X,F10.4,3X,F10.4,1X,F10.6,2X,F12.10,1X,
2          F8.6/10X,60(' ')/13X,'Properties given : ',A2,' =',
3          1PD13.6,' ',A2,' =',1PD13.6,' ')/1X,79(' ')/1X,79(' ')/)
90 FORMAT(32X,'LIST OF SYMBOLS'/32X,15(' ')/15X,'Td : Dry bulb ',
1          'temperature, ',A9/15X,'Tw : Wet bulb Temperature, ',A9/
1          15X,'Tp : Dew-point temperature, ',A9/15X,'Hu : ',
1          'Absolute humidity, fraction'/15X,'Pv : Vapor pressure',
1          ' at Td, ',A9/15X,'Ps : Saturation vapor pressure at ',
1          'Td, ',A9/15X,'Rh : Relative humidity, %'/15X,'Vh : ',
1          'Humid volume or Specific volume, ',A9/15X,'En : ',
1          'Enthalpy, ',A9/15X,'Lh : Latent heat of vaporization,',
1          1X,A9/15X,'Qh : Humid heat, ',A9/15X,'Mu : Viscosity, ',
1          A9/15X,'Ro : Density, ',A9/2X,78(' ')/)
100 FORMAT(28X,'CONVERSION TABLE'/28X,16(' ')/2X,78(' ')/)
1          2X,'[F = C*1.8 + 32; C = (F-32)/1.8]'/2X,'[1 atm = ',
1          '101325 Pa = 760 mm mercury = 14.69594878 psi = ',
1          '1033.2348 cm water]'/2X,'[1 Btu/lb = 2326.0 J/kg = ',
1          '0.55555556 kcal/kg]'/2X,'[1 Btu/lb.F = 4186.8 J/kg.C',
1          ' = 1 kcal/kg.C]'/2X,'[1 lb/ft^3 = 16.01846337 kg/m^3;',
1          ' 1 kg/m^3 = 0.06242796 lb/ft^3]'/2X,'[1 ft^3/lb = ',
1          '0.06242796 m^3/kg; 1 m^3/kg = 16.01846337 ft^3/lb]'/
1          2X,'[1 lb/ft.s = 1.488163944 Pa.s = 1488.163944 cP]'/
1          2X,78(' '))
110 FORMAT(20(/)4X,73('#')/4X,'#',19X,'THANK you for using',
1          ' this package.',19X,'#'/4X,'#',71X,'#'/4X,'#',4X,
1          'PSY2000 package has been developed by : Prof. Prabir ',
1          'K. Chandra',4X,'#'/4X,'# DCA/ESAL, 37200-000 Lavras, ',
1          'Minas Gerais, BRAZIL. FAX : (035)829-1100.#/'
1          4X,73('#')//)

RETURN
END

```

C THIS SUBROUTINE SUPPLIES VALUES TO TWO PROPERTIES

C

C

```

SUBROUTINE VALUES(L, KK)
COMMON /BLK1/ X1,X2,X3,X4,TD,TW,TP,HU,RH,VH,EN,RO,PV,SP,HL,QH,VI
COMMON /BLK2/ P0,P1,P2,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION X1,X2,X3,X4,TD,TW,TP,HU,RH,VH,EN,RO,PV,SP,HL
DOUBLE PRECISION HPV,PST,P,R,WMA,WMV,HC,CPA,CPV,CPL,QH,VI
CHARACTER *2 P0,P1,P2
CHARACTER *17 A1,A2
CHARACTER *9 A4,A5,A6,A7,A8,A9,A10
CHARACTER *20 A3

```

C

```

IF((P1 .EQ. 'HU') .OR. (P2 .EQ. 'HU')) THEN
  IF((P1 .EQ. 'PV') .OR. (P2 .EQ. 'PV')) THEN
    L = 17
    A1 = 'Absolute Humidity'
    A2 = 'Vapor Pressure'
  ENDIF
ELSEIF((P1 .EQ. 'TD') .OR. (P2 .EQ. 'TD')) THEN
  IF((P1 .EQ. 'PS') .OR. (P2 .EQ. 'PS')) THEN
    L = 18
    A1 = 'Temperature, Td'
    A2 = 'Saturn. Pressure'
  ELSEIF((P1 .EQ. 'LH') .OR. (P2 .EQ. 'LH')) THEN
    L = 19
    A1 = 'Temperature, Td'
    A2 = 'Latent Heat, Lh'
  ELSEIF((P1 .EQ. 'MU') .OR. (P2 .EQ. 'MU')) THEN
    L = 20
    A1 = 'Temperature, Td'
    A2 = 'Viscosity, Mu'
  ENDIF
ELSEIF((P1 .EQ. 'PS') .OR. (P2 .EQ. 'PS')) THEN
  IF((P1 .EQ. 'LH') .OR. (P2 .EQ. 'LH')) THEN
    L = 21
    A1 = 'Saturn. Pressure'
    A2 = 'Latent Heat, Lh'
  ELSEIF((P1 .EQ. 'MU') .OR. (P2 .EQ. 'MU')) THEN
    L = 22
    A1 = 'Saturn. Pressure'
    A2 = 'Viscosity, Mu'
  ENDIF
ELSEIF((P1 .EQ. 'LH') .OR. (P2 .EQ. 'LH')) THEN
  IF((P1 .EQ. 'MU') .OR. (P2 .EQ. 'MU')) THEN
    L = 23
    A1 = 'Latent Heat, Lh'
    A2 = 'Viscosity, Mu'
  ENDIF
ELSEIF((P1 .EQ. 'QH') .OR. (P2 .EQ. 'QH')) THEN
  IF((P1 .EQ. 'PV') .OR. (P2 .EQ. 'PV')) THEN
    L = 24

```

```

    A1 = 'Vapor Pressure'
    A2 = 'Humid Heat'
ENDIF
ENDIF
IF((P1 .EQ. 'QH') .OR. (P2 .EQ. 'QH')) THEN
    IF((P1 .EQ. 'HU') .OR. (P2 .EQ. 'HU')) THEN
        L = 25
        A1 = 'Absolute Humidity'
        A2 = 'Humid Heat'
    ENDIF
ELSEIF((P1 .EQ. 'TP') .OR. (P2 .EQ. 'TP')) THEN
    IF((P1 .EQ. 'HU') .OR. (P2 .EQ. 'HU')) THEN
        L = 26
        A1 = 'Absolute Humidity'
        A2 = 'Dew-point, Tp'
    ENDIF
ENDIF
ENDIF
IF((P1 .EQ. 'TP') .OR. (P2 .EQ. 'TP')) THEN
    IF((P1 .EQ. 'QH') .OR. (P2 .EQ. 'QH')) THEN
        L = 27
        A1 = 'Humid Heat, Qh'
        A2 = 'Dew-point, Tp'
    ELSEIF((P1 .EQ. 'PV') .OR. (P2 .EQ. 'PV')) THEN
        L = 28
        A1 = 'Vapor Pressure'
        A2 = 'Dew-point, Tp'
    ENDIF
ENDIF
ENDIF
IF((L .GE. 17) .AND. (L .LE. 28)) THEN
    RETURN
ELSE
    TD = -1.0
    TW = -1.0
    TP = -1.0
    HU = -1.0
    RH = -1.0
    VH = -1.0
    EN = -1.0
    RO = -1.0
    PV = -1.0
    SP = -1.0
    HL = -1.0
    QH = -1.0
    VI = -1.0

```

C

```

IF(P1 .EQ. 'TD') THEN
    IF(KK .EQ. 3) X1=(X3 - 32.0)/1.8
    TD = X1
ELSEIF(P1 .EQ. 'TW') THEN
    IF(KK .EQ. 3) X1=(X3 - 32.0)/1.8
    TW = X1
ELSEIF(P1 .EQ. 'HU') THEN
    HU = X1

```

```

ELSEIF(P1 .EQ. 'RH') THEN
  RH = X1/100.0
ELSEIF(P1 .EQ. 'VH') THEN
  IF(KK .EQ. 3) X1=0.06242796*X3
  VH = X1
ELSEIF(P1 .EQ. 'EN') THEN
  IF(KK .EQ. 2) THEN
    X1 = 4186.8*X3
  ELSEIF(KK .EQ. 3) THEN
    X1 = 2326.0*X3
  ENDIF
  EN = X1
ELSEIF(P1 .EQ. 'RO') THEN
  IF(KK .EQ. 3) X1=16.01846337*X3
  RO = X1
ELSEIF(P1 .EQ. 'PV') THEN
  IF(KK .EQ. 2) THEN
    X1 = 133.3223684*X3
  ELSEIF(KK .EQ. 3) THEN
    X1 = 6894.757291*X3
  ENDIF
  PV = X1
  HU = HPV(X1)
ELSEIF(P1 .EQ. 'PS') THEN
  IF(KK .EQ. 2) THEN
    X1 = 133.3223684*X3
  ELSEIF(KK .EQ. 3) THEN
    X1 = 6894.757291*X3
  ENDIF
  SP = X1
ELSEIF(P1 .EQ. 'LH') THEN
  IF(KK .EQ. 2) THEN
    X1 = 4186.8*X3
  ELSEIF(KK .EQ. 3) THEN
    X1 = 2326.0*X3
  ENDIF
  HL = X1
ELSEIF(P1 .EQ. 'MU') THEN
  IF(KK .EQ. 2) THEN
    X1 = 1.488163944*X3
  ELSEIF(KK .EQ. 3) THEN
    X1 = 1.0D-03*X3
  ENDIF
  VI = X1
  TD = 1000.0*(1.0D05*X1 - 1.7471)/4.0181
ELSEIF(P1 .EQ. 'QH') THEN
  IF((KK .EQ. 2) .OR. (KK .EQ. 3)) X1=4186.8*X3
  QH = X1
  HU = (X1 - CPA)/CPV
ELSEIF(P1 .EQ. 'TP') THEN
  IF(KK .EQ. 3) X1=(X3 - 32.0)/1.8
  TP = X1
  HU = HPV(PST(X1))

```

```

ENDIF
IF(P2 .EQ. 'TD') THEN
  IF(KK .EQ. 3) X2=(X4 - 32.0)/1.8
  TD = X2
ELSEIF(P2 .EQ. 'TW') THEN
  IF(KK .EQ. 3) X2=(X4 - 32.0)/1.8
  TW = X2
ELSEIF(P2 .EQ. 'HU') THEN
  HU = X2
ELSEIF(P2 .EQ. 'RH') THEN
  RH = X2/100.0
ELSEIF(P2 .EQ. 'VH') THEN
  IF(KK .EQ. 3) X2=0.06242796*X4
  VH = X2
ELSEIF(P2 .EQ. 'EN') THEN
  IF(KK .EQ. 2) THEN
    X2 = 4186.8*X4
  ELSEIF(KK .EQ. 3) THEN
    X2 = 2326.0*X4
  ENDIF
  EN = X2
ELSEIF(P2 .EQ. 'RO') THEN
  IF(KK .EQ. 3) X2=16.01846337*X4
  RO = X2
ELSEIF(P2 .EQ. 'PV') THEN
  IF(KK .EQ. 2) THEN
    X2 = 133.3223684*X4
  ELSEIF(KK .EQ. 3) THEN
    X2 = 6894.757291*X4
  ENDIF
  PV = X2
  HU = HPV(X2)
ELSEIF(P2 .EQ. 'PS') THEN
  IF(KK .EQ. 2) THEN
    X2 = 133.3223684*X4
  ELSEIF(KK .EQ. 3) THEN
    X2 = 6894.757291*X4
  ENDIF
  SP = X2
ELSEIF(P2 .EQ. 'LH') THEN
  IF(KK .EQ. 2) THEN
    X2 = 4186.8*X4
  ELSEIF(KK .EQ. 3) THEN
    X2 = 2326.0*X4
  ENDIF
  HL = X2
ELSEIF(P2 .EQ. 'MU') THEN
  IF(KK .EQ. 2) THEN
    X2 = 1.488163944*X4
  ELSEIF(KK .EQ. 3) THEN
    X2 = 1.0D-03*X4
  ENDIF
  VI = X2

```



```

      TD = 1000.0*(1.0D05*X2 - 1.7471)/4.0181
    ELSEIF(P2 .EQ. 'QH') THEN
      IF((KK .EQ. 2) .OR. (KK .EQ. 3)) X2=4186.8*X4
      QH = X2
      HU = (X2 - CPA)/CPV
    ELSEIF(P2 .EQ. 'TP') THEN
      IF(KK .EQ. 3) X2=(X4 - 32.0)/1.8
      TP = X2
      HU = HPV(PST(X2))
    ENDIF
  ENDIF
C
  RETURN
  END
C
C
C  THIS SUBROUTINE SELECTS TWO GIVEN PROPERTIES
C
C
C  SUBROUTINE SELECT(L,KK,M,N,T1,T2,PROP,TDS)
  COMMON /BLK1/ X1,X2,X3,X4,TD,TW,TP,HU,RH,VH,EN,RO,PV,SP,HL,QH,VI
  COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
  DOUBLE PRECISION X1,X2,X3,X4,TD,TW,TP,HU,RH,VH,EN,RO,PV,SP,HL
  DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,T1,T2,P3,QH,VI
  DOUBLE PRECISION PST,HTV,HTE,PSHRH,TDHV,TDHE,TDTHW,RH1,VH1,RO1
C
  IF(SP .GE. 0.0) THEN
    CALL TDS(SP,T1,T2,TD,M,1)
  ELSEIF(HL .GE. 0.0) THEN
    CALL TDS(HL,T1,T2,TD,M,2)
  ENDIF
  IF((RH .GE. 0.0) .AND. (VH .GE. 0.0)) THEN
    TD = (T1 + T2)/2.0
    DO 10 I=1,5000
      M = I
      HU = (P*VH - R*(TD+273.15))/((WMA/WMV)*R*(TD+273.15))
      RH1 = P*HU/PST(TD)/((WMV/WMA) + HU)
      IF(DABS(RH-RH1) .LE. 1.0D-06) GOTO 60
      TD = TD + (RH1-RH)/RH
10    CONTINUE
    ENDIF
    IF((RH .GE. 0.0) .AND. (EN .GE. 0.0)) THEN
      TD = (T1 + T2)/2.0
      DO 20 I=1,5000
        M = I
        HU = (EN - CPA*TD)/(HC + CPV*TD)
        RH1 = P*HU/PST(TD)/((WMV/WMA) + HU)
        IF(DABS(RH-RH1) .LE. 1.0D-06) GOTO 60
        TD = TD + (RH1-RH)/RH
20      CONTINUE
    ENDIF
    IF((RH .GE. 0.0) .AND. (RO .GE. 0.0)) THEN
      TD = (T1 + T2)/2.0

```

```

DO 30 I=1,5000
  M = I
  HU = (RO*R*(TD+273.15)-P)/(P-(WMA/WMV)*R*RO*(TD+273.15))
  RH1 = P*HU/PST(TD)/((WMV/WMA) + HU)
  IF(DABS(RH-RH1) .LE. 1.0D-06) GOTO 60
  TD = TD + (RH1-RH)/RH
30  CONTINUE
ENDIF
IF((VH .GE. 0.0) .AND. (EN .GE. 0.0)) THEN
  TD = (T1 + T2)/2.0
  DO 40 I=1,5000
    M = I
    HU = (EN - CPA*TD)/(HC + CPV*TD)
    VH1 = R*(TD + 273.15)*(1.0 + (WMA/WMV)*HU)/P
    IF(DABS(VH-VH1) .LE. 1.0D-05) GOTO 60
    TD = TD + (VH-VH1)/VH
40  CONTINUE
ENDIF
IF((EN .GE. 0.0) .AND. (RO .GE. 0.0)) THEN
  TD = (T1 + T2)/2.0
  DO 50 I=1,5000
    M = I
    HU = (EN - CPA*TD)/(HC + CPV*TD)
    RO1 = P*(1.0 + HU)/(R*(TD + 273.15)*(1.0 + (WMA/WMV)*HU))
    IF(DABS(RO-RO1) .LE. 1.0D-05) GOTO 60
    TD = TD + (RO1-RO)/RO
50  CONTINUE
ENDIF
60 CONTINUE
IF((TD .GE. 0.0) .AND. (TW .GE. 0.0)) THEN
  L = 1
ELSEIF((TW .GE. 0.0) .AND. (HU .GE. 0.0)) THEN
  L = 2
  TD = TDTWH(TW,HU)
ELSEIF((TD .GE. 0.0) .AND. (HU .GE. 0.0)) THEN
  L = 3
  TW = TD - 5.0
ELSEIF((TD .GE. 0.0) .AND. (RH .GE. 0.0)) THEN
  L = 4
  TW = TD - 5.0
ELSEIF((TD .GE. 0.0) .AND. (RO .GE. 0.0)) THEN
  L = 5
  TW = TD - 5.0
ELSEIF((TW .GE. 0.0) .AND. (RH .GE. 0.0)) THEN
  L = 6
  TD = TW + 5.0
ELSEIF((TW .GE. 0.0) .AND. (VH .GE. 0.0)) THEN
  L = 7
  TD = TW + 5.0
ELSEIF((TW .GE. 0.0) .AND. (EN .GE. 0.0)) THEN
  L = 8
  TD = TW + 5.0
ELSEIF((TW .GE. 0.0) .AND. (RO .GE. 0.0)) THEN

```

```

      L = 9
      TD = TW + 5.0
      ELSEIF((TD .GE. 0.0) .AND. (VH .GE. 0.0)) THEN
        L = 10
        HU = HTV(TD,VH)
      ELSEIF((TD .GE. 0.0) .AND. (EN .GE. 0.0)) THEN
        L = 11
        HU = HTE(TD,EN)
      ELSEIF((HU .GE. 0.0) .AND. (RH .GE. 0.0)) THEN
        L = 12
        SP = PSHRH(HU,RH)
        CALL TDS(SP,T1,T2,TD,M,1)
      ELSEIF((HU .GE. 0.0) .AND. (VH .GE. 0.0)) THEN
        L = 13
        TD = TDHV(HU,VH)
      ELSEIF((HU .GE. 0.0) .AND. (EN .GE. 0.0)) THEN
        L = 14
        TD = TDHE(HU,EN)
      ELSEIF((HU .GE. 0.0) .AND. (RO .GE. 0.0)) THEN
        L = 15
        VH = (1.0 + HU)/RO
        TD = TDHV(HU,VH)
      ELSEIF((VH .GE. 0.0) .AND. (RO .GE. 0.0)) THEN
        L = 16
        HU = VH*RO - 1.0
        TD = TDHV(HU,VH)
      ENDIF
      IF(M .EQ. 5000) RETURN
      CALL PROP(L,KK,M,N,T1,T2)
C
      RETURN
      END
C
C
C
C
C
C
      SUBROUTINE PROP(L,KK,M,N,T1,T2)
      COMMON /BLK1/ X1,X2,X3,X4,TD,TW,TP,HU,RH,VH,EN,RO,PV,SP,HL,QH,VI
      COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
      DOUBLE PRECISION X1,X2,X3,X4,TD,TW,TP,HU,RH,VH,EN,RO,PV,SP,HL
      DOUBLE PRECISION P,R,WMA,WMV,HC,CPA,CPV,CPL,T1,T2,P3,QH,VI
      DOUBLE PRECISION HDBWB,RHPVT,ROTH,VTH,PVH,ENTH,PST,HQH,VST,HLT
      DOUBLE PRECISION RHHT,HU1,RH1,RO1,VH1,EN1,PV1
C
      IF(L .EQ. 3) THEN
        DO 10 I=1,5000
          N = I
          HU1 = HDBWB(TD,TW)
          IF(DABS(HU-HU1) .LE. 1.0D-06) GOTO 70
          TW = TW + (HU-HU1)/HU
10    CONTINUE
        ELSEIF((L .EQ. 4) .OR. (L .EQ. 6)) THEN

```

```

DO 20 I=1,5000
  N = I
  RH1 = RHPVT(PVH(HDBWB(TD,TW)),TD)
  IF(DABS(RH-RH1) .LE. 1.0D-06) GOTO 70
  IF(L .EQ. 4) THEN
    TW = TW + (RH-RH1)/RH
  ELSE
    TD = TD + (RH1-RH)/RH
  ENDIF
20 CONTINUE
  ELSEIF((L .EQ. 5) .OR. (L .EQ. 9)) THEN
    DO 30 I=1,5000
      N = I
      RO1 = ROTH(TD,HDBWB(TD,TW))
      IF(DABS(RO-RO1) .LE. 1.0D-04) GOTO 70
      IF(L .EQ. 5) THEN
        TW = TW + (RO1-RO)/RO
      ELSE
        TD = TD + (RO1-RO)/RO
      ENDIF
30 CONTINUE
    ELSEIF(L .EQ. 7) THEN
      DO 40 I=1,5000
        N = I
        VH1 = VTH(TD,HDBWB(TD,TW))
        IF(DABS(VH-VH1) .LE. 1.0D-04) GOTO 70
        TD = TD + (VH-VH1)/VH
40 CONTINUE
    ELSEIF(L .EQ. 8) THEN
      DO 50 I=1,5000
        N = I
        EN1 = ENTH(TD,HDBWB(TD,TW))
        IF(DABS(EN-EN1) .LE. 0.05) GOTO 70
        TD = TD + (EN1-EN)/EN
50 CONTINUE
    ELSEIF((L .GE. 10) .AND. (L .LE. 16)) THEN
      TW = TD - 5.0
      DO 60 I=1,5000
        N = I
        HU1 = HDBWB(TD,TW)
        IF(DABS(HU-HU1) .LE. 1.0D-06) GOTO 70
        TW = TW + (HU-HU1)/HU
60 CONTINUE
      ENDIF
C
70 HU = HDBWB(TD,TW)
  PV = PVH(HU)
  TP = TW - 5.0
  DO 80 I=1,5000
    N = I
    PV1 = PST(TP)
    IF(DABS(PV-PV1) .LE. 0.05) GOTO 90
    TP = TP + (PV - PV1)/PV

```

```

80 CONTINUE
C
90 RO = ROTH(TD,HU)
  RH = RHHT(HU,TD)
  VH = VTH(TD,HU)
  EN = ENTH(TD,HU)
  HL = HLT(TD)
  SP = PST(TD)
  QH = HQH(HU)
  VI = VST(TD)
  IF(KK .EQ. 2) THEN
    EN = EN/4186.8
    HL = HL/4186.8
    SP = SP/133.3223684
    PV = PV/133.3223684
    QH = QH/4186.8
    VI = 1000.0*VI
  ELSEIF(KK .EQ. 3) THEN
    TD = TD*1.8 + 32.0
    TW = TW*1.8 + 32.0
    TP = TP*1.8 + 32.0
    VH = 16.01846337*VH
    EN = EN/2326.0
    HL = HL/2326.0
    RO = 0.06242796*RO
    SP = SP/6894.757291
    PV = PV/6894.757291
    QH = QH/4186.8
    VI = VI/1.488163944
  ENDIF
C
  RETURN
END
C
C
C THIS SUBROUTINE CALCULATES TEMPERATURE BY THE METHOD OF CONVERGENCE
C
C
SUBROUTINE TDS(S,T1,T2,TD1,M,I)
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION PST,HLT,P,R,WMA,WMV,HC,CPA,CPV,CPL,S,T1,T2,TD1
DOUBLE PRECISION A,B,C,D,S1,ERROR
C
  IF(I .EQ. 1) THEN
    C = PST(T2)
    D = PST(T1)
  ELSEIF(I .EQ. 2) THEN
    C = HLT(T2)
    D = HLT(T1)
  ENDIF
  A = (C*T1 - D*T2)/(T2 - T1)
  B = (A + S)/2.0
  TD1 = B*(T1/(A+D) + T2/(A+C))

```

```

C
DO 10 J=1,5000
  M = J
  IF(I .EQ. 1) THEN
    S1 = PST(TD1)
    ERROR = 1.0D-02
  ELSEIF(I .EQ. 2) THEN
    S1 = HLT(TD1)
    ERROR = 0.05
  ENDIF
  IF(DABS(S-S1) .LE. ERROR) RETURN
  TD1 = TD1 + (S-S1)/S
10 CONTINUE
C
  RETURN
END
C
C
C THIS "PSYCRO.FOR" HAS ALL THE NECESSARY PSYCHROMETRIC
C FUNCTIONS [LISTED IN APPENDIX A.5.1]
C
C
C $INCLUDE:'PSYCRO.FOR'
C

```

A.5.3. Thermal properties for the fin problem listed in Appendix A.5.4.
 Program name : **THERMO.FOR**

```

C
C
C THERMOPHYSICAL PROPERTIES OF MATERIAL
C
C
C
C RO = 8825.0
C CP = 390.0
C XK = 40.0
C

```

A.5.4. Complete program listing for the circular fin problem.
 Program name : **FIN.FOR**

```

C
C THIS PROGRAM COUPLED WITH THE SUBROUTINES 'COEFF' AND
C "TRIDGL" CAN SOLVE A FIN PROBLEM BY THE IMPLICIT SCHEME.
C
C
C Program Developed by: Prof. Prabir K. Chandra

```

```

C
C
COMMON /BLOK1/ TIME, TTIME, DELT, DD, DD1, TC, TB, T0
COMMON /BLOK2/ A(101), B(101), D(101), R(101), V(101), T(1, 101),
1      TS(101), AR(101), V0, W, DELR, N
COMMON /PROP/ RO, ALFA, CP, XK, H, PI
DOUBLE PRECISION TIME, TTIME, DELR, DELT, DD, DD1, TC, T0, TB, RO, ALFA,
1      CP, XK, H, V0, R1, R2, W, TMEAN, PI, DT, Q, ER
DOUBLE PRECISION A, B, D, R, V, T, TS, AR
CHARACTER S *15
CHARACTER S1 *20, S0 *1

C
C  DELR, DELT, TIME, TTIME : STEP SIZE IN SPACE (m) AND TIME (s),
C  VARIABLE AND TOTAL TIME OF EXPOSURE (s), RESPECTIVELY.
C  D, A, B and N: DIAGONAL, UPPER, LOWER ELEMENTS and ROW OR
C  COLUMN SIZE OF TRIANGULAR MATRIX.
C  R, T: ELEMENTS OF CONSTANT VECTOR AND SOLUTION VECTOR.
C  RO, ALFA, CP, XK, H : DENSITY (kg/m^3), DIFFUSIVITY (m^2/s),
C  SPECIFIC HEAT (J/kg.C), CONDUCTIVITY (W/m.C) AND HEAT
C  TRANSFER COEFFICIENT (W/m^2.C).
C  AR(J), V(J) : J-TH AREA PERPENDICULAR TO FLOW (m^2) AND
C  SEGMENT VOLUME (m^3), RESPECTIVELY.
C  TC, T0 : TEMPERATURE AT SIDE 1 AND SIDE 2 AND INITIAL (C).
C
C  OPEN(20, FILE='DATA.OUT', STATUS='NEW')

C
C  DEFINE THERMO-PHYSICAL PROPERTIES ...
C
$INCLUDE:'THERMO.FOR'
C
ER = 1.0D-03
PI = 3.141592654
ALFA = XK/RO/CP
S = 'IMPLICIT SCHEME'
DO 70 II=1, 100
    WRITE(*, 80) S
    READ(*, 90) S0
    IF((S0.EQ. 'N') .OR. (S0.EQ. 'n')) THEN
        CLOSE(20, STATUS='KEEP')
        STOP
    ENDIF
    L = 2
    S1 = 'CYLINDRICAL GEOMETRY'
    WRITE(*, 100)
    READ(*, *) T0, TB, TC, H
    WRITE(*, 110)
    READ(*, *) R1, R2, W, TTIME, N
    R1 = R1/100.0
    R2 = R2/100.0
    W = W/100.0
    DELR = (R2 - R1)/N

C
C  AREAS AND VOLUMES OF CYLINDRICAL FIN SEGMENTS ARE CALCULATED ...

```

```

C
DO 10 J=1,N+1
    AR(J) = 2.0*PI*(R2 - (J-1)*DEL R)*W
    V(J) = PI*DEL R*(2.0*R2 - (2*J-1)*DEL R)*W
10  CONTINUE
    V0 = PI*(R2**2 - R1**2)*W

C
    DELT = 0.5*DEL R*DEL R/ALFA
    WRITE(*,120) DELT
    READ(*,*) DELT
    NTIME = TTIME/DELT
    DD = -DEL R*DEL R/ALFA/DELT
    DD1 = XK*(AR(1) + AR(2))/(2.0*AR(1)*H*DEL R)

C
C  TIME IS INITIALIZED ... INITIAL CONDITION IS ATTRIBUTED
C
    TIME = 0.0
    DO 20 J=1,N+1
        T(1,J) = T0
20  CONTINUE
C
    WRITE(20,130) S,S1
    WRITE(*,140)
    READ(*,*) KR
    IF(KR .EQ. 1) THEN
        WRITE(*,150) NTIME
        READ(*,*) INTV
    ELSEIF(KR .EQ. 2) THEN
        INTV = NTIME
    ELSE
        STOP
    ENDIF
    WRITE(*,160) S,S1

C
C  INITIAL VALUES ARE PRINTED ...
C
    TMEAN = 0.0
    DO 30 J=1,N
        TMEAN = TMEAN + V(J)*(T(1,J) + T(1,J+1))/2.0/V0
30  CONTINUE
    WRITE(*,210) TMEAN,TIME,(T(1,J),J=1,N+1)
    WRITE(20,210) TMEAN,TIME,(T(1,J),J=1,N+1)

C
    IK = 0

C
    T(1,1) = (TC + DD1*T(1,2))/(1.0 + DD1)
    T(1,N+1) = TB

C
C  TIME IS INCREMENTED BY DELT AND BOUNDARY VALUES ARE GIVEN ...
C
40  TIME = TIME + DELT
    IK = IK + 1

C

```



```

CALL COEFF
CALL TRIDGL(N)
C
TS(1) = (TC + DD1*TS(2))/(1.0 + DD1)
TS(N+1) = TB
C
TMEAN = 0.0
DO 50 J=1,N
    TMEAN = TMEAN + V(J)*(TS(J) + TS(J+1))/2.0/V0
50  CONTINUE
C
C  STEADY STATE CONDITION IS CHECKED ...
C
    IF(IK .GT. 2) THEN
        DT = DABS(T(1,N) - TS(N))
        IF((DT .LE. ER) .AND. (TIME .LE. TTIME)) THEN
            Q = -XK*(AR(N) + AR(N+1))*(TS(N) - TS(N+1))/DELR/2.0
            WRITE(*,170) TMEAN,TIME,(TS(J),J=1,N+1)
            WRITE(20,180) TMEAN,TIME,(TS(J),J=1,N+1)
            WRITE(*,190) Q
            WRITE(20,190) Q
            PAUSE 'Going to the MAIN MENU ...'
            GOTO 70
        ELSEIF(TIME .GE. TTIME) THEN
            WRITE(*,200) TIME
            PAUSE 'Going to the MAIN MENU ...'
            GOTO 70
        ENDIF
    ENDIF
C
C  OLD VALUES ARE GIVEN NEW VALUES FOR THE NEXT TIMESTEP ...
C
    DO 60 J=1,N+1
        T(1,J) = TS(J)
60  CONTINUE
C
    IK1 = (IK/INTV)*INTV
    IF(IK1 .EQ. IK) THEN
        WRITE(*,210) TMEAN,TIME,(T(1,J),J=1,N+1)
        WRITE(20,210) TMEAN,TIME,(T(1,J),J=1,N+1)
        IF(TIME .LT. TTIME) THEN
            PAUSE ' '
        ELSE
            WRITE(*,*)
            PAUSE 'Going to the MAIN MENU ...'
            GOTO 70
        ENDIF
    ENDIF
    GOTO 40
70  CONTINUE
C
C  *****  FORMAT STATEMENTS  *****
C

```

```

80 FORMAT(32X,A15/32X,15('-')/20X,39('-')/21X,'Press [Y]',
1      ' to CONTINUE and [N] to QUIT'/20X,39('-')/)
90 FORMAT(A1)
100 FORMAT(20(/)14X,53('-')//15X,'Give INITIAL, BASE and OUTSIDE ',
1      'temperatures [C] and'/17X,'HEAT TRANSFER COEFFICIENT',
2      '[W/m^2.C] and <ENTER>'//14X,53('-')//)
110 FORMAT(20(/)12X,55('-')/13X,'Give base RADIUS, fin RADIUS and',
1      ' fin WIDTH in [cm] and'/18X,'TOTAL time [s], No. of ',
2      'DIVISIONS and <ENTER>'//18X,'[Number of DIVISIONS ',
3      'SHOULD NOT exceed 100]'/12X,55('-')//)
120 FORMAT(20(/)18X,41('-')/23X,'Give TIME step in [s] and <ENTER>.'
1      '/19X,'[Lower limit of TIME step = 'F10.5,']'/18X,41('-')
1      //)
130 FORMAT(/21X,A15, ' - ',A20/21X,38('-')/)
140 FORMAT(20(/)19X,42('-')/20X,'Type [1] : to PRINT step-by-step',
1      ' RESULTS'/20X,'Type [2] : to PRINT final RESULTS only.'
1      '/20X,'Type [3] : to STOP'/19X,42('-')//)
150 FORMAT(20(/)5X,71('-')/6X,'Give PRINTING INTERVAL [No. of ',
1      'TIME steps to be skipped] and <ENTER>'//25X,'[It ',
1      'SHOULD be LESS than '15,']'/5X,71('-')//)
160 FORMAT(20(/)21X,A15, ' - ',A20/21X,38('-')/)
170 FORMAT(20(/)15X,50('-')/32X,'At STEADY STATE'/32X,15('-')//16X,
1      'Mean Temperature = 'F6.2, ' C at Time = 'F8.2, ' s'/15X,
2      '50('')/(16X,6(F6.2,2X)))
180 FORMAT(/15X,50('-')/32X,'At STEADY STATE'/32X,15('-')//16X,
1      'Mean Temperature = 'F6.2, ' C at Time = 'F8.2, ' s'/15X,
2      '50('')/(16X,6(F6.2,2X)))
190 FORMAT(/19X,'[Rate of HEAT dissipated = 'F11.5, ' W]'/5X,
1      '71('')//)
200 FORMAT(20(/)11X,56('-')/13X,'STEADY STATE needs MORE THAN ',
1      'F10.4, ' s to reach ...'/11X,56('-')//)
210 FORMAT(/16X,'Mean Temperature = 'F6.2, ' C at Time = 'F8.2,
1      ' s'/15X,50('-')/(16X,6(F6.2,2X)))
      STOP
      END

```

```

C
C
C THIS SUBROUTINE SUPPLIES COEFFICIENT VALUES TO "TRIDGL" ...
C
C
SUBROUTINE COEFF
COMMON /BLOK1/ TIME,TTIME,DELT,DD,DD1,TC1,TC2,T0
COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
1      TS(101),AR(101),V0,X1,DELR,N
COMMON /PROP/ RO,ALFA,CP,XK,H,PI
DOUBLE PRECISION TIME,TTIME,DELT,DD,DD1,TC1,TC2,T0,C1,C2,C3,CC0
DOUBLE PRECISION A,B,D,R,V,T,TS,AR,V0,X1,DELR,CC,S
DOUBLE PRECISION RO,ALFA,CP,XK,H,PI

```

```

C
C ELEMENTS OF TRIDIAGONAL MATRIX AND CONSTANT VECTOR ARE DEFINED
C
      S = 2.0*H*DELT*DD/RO/CP/X1
      DO 10 J=2,N

```

```

C1 = (AR(J) + AR(J-1))/2.0/AR(J)
C3 = (AR(J+1) + AR(J))/2.0/AR(J)
C2 = DD - C1 - C3 + S
IF(J .EQ. 2) THEN
  CC0 = C1*DD1/(1.0 + DD1)
  CC = C1*TC1/(1.0 + DD1)
ENDIF
B(J) = C1
D(J) = C2
IF(J .EQ. 2) D(J) = C2 + CC0
A(J) = C3
R(J) = DD*T(1,J) + S*TC1
10 CONTINUE
C
C THE FIRST AND THE LAST ELEMENTS OF THE CONSTANT VECTOR ARE
C CORRECTED ACCORDING TO DIFFERENT BOUNDARY CONDITIONS ...
C
  R(2) = R(2) - CC
  R(N) = R(N) - C3*TC2
C
  RETURN
END
C
C
C SUBROUTINE TRIDGL SOLVES 1-D PARABOLIC PDE USING TRIDIAGONAL MATRIX.
C [LISTED IN APPENDIX A.3.7.3]
C
C
C $INCLUDE:'TRIDGL.FOR'
C

```

A.5.5. Thermal properties for the puffing problem (Appendix A.5.7).
 Program name : **THERMO1.FOR**

```

C
C
C THERMOPHYSICAL PROPERTIES OF THE PURE WATER AND THE
C PRODUCT
C
C
C ROW(T) GIVES DENSITY OF WATER AS A FUNCTION OF TEMPERATURE
C
C
C DOUBLE PRECISION FUNCTION ROW(T)
C DOUBLE PRECISION T
C
  ROW = 0.1001404D04 - 0.1205817*T - 0.3204874D-02*T*T +
1      0.1753593D-05*T**3
  RETURN
END

```

```

C
C
C ROP(T,XM) GIVES DENSITY OF PRODUCT AS A FUNCTION OF
C TEMPERATURE AND MOISTURE, kg/m^3
C
C
C DOUBLE PRECISION FUNCTION ROP(T,XM)
C DOUBLE PRECISION ROW,T,XM
C
C ROP = 920.0
C RETURN
C END
C
C
C CPW(T) GIVES SP. HEAT OF WATER AS A FUNCTION OF TEMPERATURE
C
C
C DOUBLE PRECISION FUNCTION CPW(T)
C DOUBLE PRECISION T
C
C CPW = 0.4202864D02 - 0.6594387*T + 0.2290745D-02*T*T +
C 0.4206725D-04*T**3
C RETURN
C END
C
C
C CPP(T,XM) GIVES SP. HEAT OF PRODUCT AS A FUNCTION OF
C TEMPERATURE AND MOISTURE, J/kg.C
C
C
C DOUBLE PRECISION FUNCTION CPP(T,XM)
C DOUBLE PRECISION CPW,T,XM
C
C CPP = 2753.18
C RETURN
C END
C
C
C KW(T) GIVES THERMAL CONDUCTIVITY OF WATER AS A FUNCTION OF
C TEMPERATURE, W/m.C
C
C
C DOUBLE PRECISION FUNCTION KW(T)
C DOUBLE PRECISION T
C
C KW = 0.5540917 + 0.258017D-02*T - 0.1717135D-04*T**2 +
C 0.4838772D-07*T**3 - 0.6704172D-10*T**4
C RETURN
C END
C
C
C KP(T,XM) GIVES THERMAL CONDUCTIVITY OF PRODUCT AS A
C FUNCTION OF TEMPERATURE AND MOISTURE, W/m.C

```

```

C _____
C
  DOUBLE PRECISION FUNCTION KP(T,XM)
  DOUBLE PRECISION KW,T,XM

C
  KP = 0.15*(1.0 - XM) + KW(T)*XM
  RETURN
  END

C _____
C
C ALFAP(T,XM) GIVES THERMAL DIFFUSIVITY OF PRODUCT AS A
C FUNCTION OF TEMPERATURE AND MOISTURE, m^2/s
C _____
C
  DOUBLE PRECISION FUNCTION ALFAP(T,XM)
  DOUBLE PRECISION KP,ROP,CPP,T,XM

C
  ALFAP = KP(T,XM)/ROP(T,XM)/CPP(T,XM)
  RETURN
  END

C _____

```

A.5.6. Program segment "COEFF.FOR" for supplying coefficient values to "TRIDGL.FOR" in "PUFF.FOR" and in "DRY2000.FOR".

Program name : **COEFF.FOR**

```

C _____
C
C ...THIS SUBROUTINE CALCULATES THE COEFFICIENTS FOR SUBROUTINE TRIDGL
C _____
C
  SUBROUTINE COEFF
  COMMON /BLOK1/ TIME,TTIME,DELT,DD,DD1,TC,T0
  COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
  I          TS(101),AR(101),V0,X1,DELR,N
  DOUBLE PRECISION TIME,TTIME,DELT,DD,DD1,TC,T0,C1,C2,C3,CC0
  DOUBLE PRECISION A,B,D,R,V,T,TS,AR,V0,X1,DELR,CC

C
C ELEMENTS OF TRIDIAGONAL MATRIX AND CONSTANT VECTOR ARE DEFINED
C
  DO 10 J=2,N
    C1 = (AR(J) + AR(J-1))/2.0/AR(J)
    C3 = (AR(J+1) + AR(J))/2.0/AR(J)
    C2 = DD - C1 - C3

```

```

      IF(J .EQ. 2) THEN
        CC0 = C1*DD1/(1.0 + DD1)
        CC = C1*TC/(1.0 + DD1)
      ENDIF
      B(J) = C1
      D(J) = C2
      IF(J .EQ. 2) D(J) = C2 + CC0
      IF(J .EQ. N) D(J) = C2 + C3
      A(J) = C3
      R(J) = DD*T(1,J)
10  CONTINUE
C
C  THE FIRST AND THE LAST ELEMENTS OF THE CONSTANT VECTOR ARE
C  CORRECTED ACCORDING TO DIFFERENT BOUNDARY CONDITIONS ...
C
      R(2) = R(2) - CC
C
      RETURN
      END
C

```

A.5.7. Complete program listing for the calculation of heat transfer coefficient for the puffing of rice by heated sand.

Program name : **PUFF.FOR**

```

C
C
C  THIS PROGRAM COUPLED WITH THE SUBROUTINES 'COEFF' AND "TRIDGL"
C  CAN CALCULATE THE VALUE OF HEAT TRANSFER COEFFICIENT OF SAND
C  PUFFING OF RICE BY AN ITERATIVE METHOD.
C
C  Program Developed by: Prof. Prabir K. Chandra
C
C
C
COMMON /BLOK1/ TIME,TTIME,DELT,DD,DD1,TC,T0
COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),T(1,101),
1             TS(101),AR(101),V0,X1,DELR,N
COMMON /PROP/ RO,ALFA,CP,XK,H,PI
DOUBLE PRECISION TIME,TTIME,DELT,DD,DD1,TC,T0,RO,ALFA,CP,XK,
1             H,PI,TMEAN,XM
DOUBLE PRECISION A,B,D,R,V,T,TS,AR,V0,X1,DELR,TAV,H0,ERROR
DOUBLE PRECISION ROW,ROP,CPW,CP,KW,KP,ALFAP
CHARACTER S *15, S2 *52
CHARACTER S1 *20, S0 *1

```

```

C
C  DELR, DELT, TIME, TTIME : STEP SIZE IN SPACE [m] AND TIME [s],
C    VARIABLE AND TOTAL TIME OF EXPOSURE [s], RESPECTIVELY.
C  D, A, B and N: DIAGONAL, UPPER, LOWER ELEMENTS and ROW OR
C    COLUMN SIZE OF TRIANGULAR MATRIX.
C  R, T: ELEMENTS OF CONSTANT VECTOR AND SOLUTION VECTOR.
C  RO, ALFA, CP, XK, H : DENSITY [kg/m^3], DIFFUSIVITY [m^2/s],
C    SPECIFIC HEAT [J/kg.C], CONDUCTIVITY [W/m.C] AND HEAT
C    TRANSFER COEFFICIENT [W/m^2.C].
C  AR(J), V(J) : J-TH AREA PERPENDICULAR TO FLOW [m^2] AND
C    SEGMENT VOLUME [m^3], RESPECTIVELY.
C  TC,T0 : TEMPERATURE AT SIDE 1 AND SIDE 2 AND INITIAL [C].
C  X1,XM : THICKNESS/RADIUS [m] AND MOISTURE CONTENT OF THE
C    PRODUCT [fraction w.b.].
C
C    OPEN(20,FILE='DATA.OUT',STATUS='NEW')
C
C  DEFINE THERMO-PHYSICAL PROPERTIES ...
C
C    PI = 3.141592654
C    ERROR = 1.0D-02
C    S = 'IMPLICIT SCHEME'
C    DO 80 II=1,100
C      WRITE(*,90) S
C      READ(*,100) S0
C      IF((S0.EQ. 'N') .OR. (S0.EQ. 'n')) THEN
C        CLOSE(20,STATUS='KEEP')
C        STOP
C      ENDIF
C      WRITE(*,110) S
C      READ(*,*) L
C      IF(L.EQ. 1) THEN
C        S1 = 'RECTANGULAR GEOMETRY'
C      ELSEIF(L.EQ. 2) THEN
C        S1 = 'CYLINDRICAL GEOMETRY'
C      ELSEIF(L.EQ. 3) THEN
C        S1 = 'SPHERICAL GEOMETRY'
C      ELSE
C        GOTO 80
C      ENDIF
C      S2 = 'FREE CONVECTIVE BOUNDARY'
C      WRITE(*,120) S2
C      READ(*,*) T0,TC,TAV,XM,H0
C      WRITE(*,130)
C      READ(*,*) X1,TTIME,N
C      X1 = X1/1000.0
C      XM = XM/100.0
C      IF(L.EQ. 1) THEN
C        WRITE(*,140)
C        PAUSE ''
C        WRITE(20,150)
C        DELR = X1/2.0/N
C      ELSE

```

```

        DELR = X1/N
    ENDIF
C
    CALL GEOMTY(L)
C
    TMEAN = TAV
    ALFA = ALFAP(TMEAN,XM)
    DELT = 0.5*DELR*DELR/ALFA
    WRITE(*,160) DELT
    READ(*,*) DELT
    H = H0
    DD = -DELR*DELR/ALFA/DELT
    XK = KP(TMEAN,XM)
    DD1 = XK*(AR(1) + AR(2))/(2.0*AR(1)*H*DELR)
    WRITE(20,170) S,S1,S2
    WRITE(*,180) S,S1,S2
    M = 0
C
C  TIME IS INITIALIZED ... INITIAL CONDITION IS ATTRIBUTED
C
30    TIME = 0.0
    IF(M .GE. 100) THEN
        WRITE(*,190)
        PAUSE ' '
        GOTO 80
    ENDIF
    M = M + 1
    WRITE(*,200) M
    DO 40 J=1,N+1
        T(1,J) = T0
40    CONTINUE
C
C  BOUNDARY CONDITIONS ARE SPECIFIED ...
C
    T(1,1) = (TC + DD1*T(1,2))/(1.0 + DD1)
    T(1,N+1) = T(1,N)
C
C  TIME IS INCREMENTED BY DELT AND BOUNDARY VALUES ARE GIVEN ...
C
50    TIME = TIME + DELT
C
    CALL COEFF
    CALL TRIDGL(N)
C
    TS(1) = (TC + DD1*TS(2))/(1.0 + DD1)
    TS(N+1) = TS(N)
C
    TMEAN = 0.0
    DO 60 J=1,N
        TMEAN = TMEAN + V(J)*(TS(J) + TS(J+1))/2.0/V0
60    CONTINUE
C
    ALFA = ALFAP(TMEAN,XM)

```



```

DD = -DEL R*DEL R/ALFA/DEL T
XK = KP(TMEAN,XM)
DD1 = XK*(AR(1) + AR(2))/(2.0*AR(1)*H*DEL R)
C
C OLD VALUES ARE GIVEN NEW VALUES FOR THE NEXT TIMESTEP ...
C
DO 70 J=1,N+1
    T(1,J) = TS(J)
70 CONTINUE
C
    IF (TIME .GE. TTIME) THEN
C
C HEAT TRANSFER COEFFICIENT VALUE IS CORRECTED IF TMEAN-COMPUTED
C DOES NOT DIFFER FROM TMEAN-MEASURED BY A GIVEN SMALL
DIFFERENCE
C
        IF (DABS(TMEAN - TAV) .LE. ERROR) THEN
            WRITE(*,210) TMEAN,H,M
            WRITE(20,210) TMEAN,H,M
            WRITE(*,*)
            PAUSE 'Going to MAIN MENU ...'
            GOTO 80
        ENDIF
        H = H0 + TAV - TMEAN
        H0 = H
        GOTO 30
    ENDIF
    GOTO 50
80 CONTINUE
C
C ***** FORMAT STATEMENTS *****
C
90 FORMAT(25(/)32X,A15/32X,15('-')/20X,39('-')//21X,'Press [Y]',
1      ' to CONTINUE and [N] to QUIT'/20X,39('-')//)
100 FORMAT(A1)
110 FORMAT(25(/)1X,79('#')/1X,'#',77X,'#'/1X,'#',18X,'Solution ',
1      'of Partial Differential Equation',18X,'#'/1X,'#',18X,
1      41('-'),18X,'#'/1X,'#',77X,'#'/1X,'#',31X,A15,31X,'#'/
1      1X,'#',77X,'#'/1X,'#',24X,'DECLARE PRODUCT CONFIGURA',
1      'TION',24X,'#'/1X,'#',24X,29('-'),24X,'#'/1X,'#',77X,
1      '#'/1X,'#',31X,'1: RECTANGULAR',32X,'#'/1X,'#',31X,
1      '2: CYLINDRICAL',32X,'#'/1X,'#',31X,'3: SPHERICAL',34X,
1      '#'/1X,'#',77X,'#'/1X,'#',13X,'[Type any OTHER number',
1      ' to RETURN to the BEGINNING]',14X,'#'/1X,'#',77X,'#'/
1      1X,79('#')//)
120 FORMAT(25(/)3X,74('-')/14X,A52/14X,52('-')//4X,'Give ',
1      'INITIAL, OUTSIDE and AVERAGE temperatures [C], MOIST',
1      'URE [% w.b] and/7X,'the First GUESS of HEAT TRANSFER',
1      ' COEFFICIENT [W/m^2.C] and <ENTER>/3X,74('-')//)
130 FORMAT(25(/)1X,78('-')/3X,'Give THICKNESS/RADIUS in [mm]. ',
1      'TOTAL time [s], No. of DIVISIONS and <ENTER>'/1X,
1      78('-')//)
140 FORMAT(25(/)10X,60('-')/12X,'Due to SYMMETRY only HALF of the',

```

```

1          ' geometry will be SOLVED.'/10X,60('-')//)
150 FORMAT(/10X,60('-')/12X,'Due to SYMMETRY only HALF of the',
1          ' geometry will be SOLVED.'/10X,60('-'))
160 FORMAT(25(/)18X,41('-')/23X,'Give TIME step in [s] and <ENTER>'/
1          19x,'[Lower limit of TIME step = ',F10.5,']'/18X,41('-')
1          //)
170 FORMAT(/21X,A15,' - ',A20/21X,38('-')/13X,['A52,']//)
180 FORMAT(20(/)21X,A15,' - ',A20/21X,38('-')/13X,['A52,']//)
1          36X,'Wait ...'//)
190 FORMAT(25(/)16X,'Scheme is NOT converging ... GOING to MAIN ',
1          'MENU'//)
200 FORMAT(37X,['I3,']//)
210 FORMAT(7X,65('-')/8X,'Mean Temperature = ',F7.2,' C and ',
1          'ESTIMATED h = ',F8.4,' W/m^2.C'/18X,['Number of itera',
2          'tions for CONVERGENCE = ',I3,']'/7X,65('-'))
      STOP
      END
C _____
C
C THIS SUBROUTINE CALCULATES AREA AND VOLUME OF DIFFERENT SHAPES
C [INCLUDED IN APPENDIX A.3.7.2]
C _____
C
$INCLUDE:'GEOMTY.FOR'
C _____
C
C SUBROUTINE COEFF CALCULATES COEFFICIENTS FOR SUBROUTINE TRIDGL
C _____
C
$INCLUDE:'COEFF.FOR'
C _____
C
C _____
C
C THIS SUBROUTINE SOLVES A 1-D PARABOLIC PDE USING TRIDIAGONAL
C MATRIX. [LISTED IN APPENDIX A.3.7.3]
C _____
C
$INCLUDE:'TRIDGL.FOR'
C _____
C
C THE FOLLOWING FUNCTIONS DEFINES THERMOPHYSICAL PROPERTIES OF
C THE PURE WATER AND THE PRODUCT
C
$INCLUDE:'THERMO1.FOR'
C _____

```

A.5.8. Complete program listing for the cyclone separator.

Program name : **CYCLONE.FOR**

```

C _____

```

```

C
C THIS PROGRAM TRACES THE PATH OF A CYCLONE AND DETERMINES
C TIME OF SETTLEMENT OF A PARTICLE MOVING INSIDE IT.
C
C
COMMON /BLOK3/ Y(10,5,4),Z(10,5,4),Y0(10,4),YT(10,101),TT,ER1
COMMON /BLOK4/ FVALUE,TIVALU,DELT,NEQ,NORDER,K,L
COMMON /BLOK5/ C1,C2
COMMON /PSYCR/ P,R,WMA,WMV,HC,CPA,CPV,CPL
DOUBLE PRECISION Y,Z,Y0,YT,TT,ER1,FVALUE,TIVALU,DELT,P,C1,C2,WMA
DOUBLE PRECISION WMV,HC,CPA,CPV,CPL,D1,D2,T0,TW,AFLO,AFAN,DP,PI
DOUBLE PRECISION ROP,ROA,HU,VAIR,OMEGA,R1,R2,RP,R,XL,FW,FH
DOUBLE PRECISION F,PST,VTH,ROTH,HDBWB,HPV,VST
CHARACTER S *35, S1 *1
EXTERNAL F

C
C KK : INDEX NUMBER - [1] FOR R-K; [2] FOR ADAMS P-C and
C [3] FOR HAMMING'S P-C METHODS
C NEQ, DELT : NUMBER OF EQUATIONS AND STEPSIZE.
C NORDER : ORDER OF 'NEQ' DIFFERENTIAL EQUATIONS
C TIVALU, FVALUE : INITIAL & FINAL VALUES OF INDEPENDENT VAR.
C D1,D2 : THROAT AND INSIDE DIAMETER OF CYCLONE, m
C T0,TW : DRY AND WET BULB TEMPERATURE OF AMBIENT AIR, C
C AFLO : AIR FLOW RATE THROUGH THE CYCLONE INLET, m^3/s
C AFAN,DP : AREA OF FAN INLET [m^2] and PARTICLE DIAMETER, micron
C WMA,WMV : MOLECULAR WEIGHT OF AIR AND WATER
C HC : HEAT CAPACITY OF AIR, J/kg
C CPA,CPV,CPL : SPECIFIC HEAT OF AIR, WATER VAPOR AND LIQUID
C WATER, J/kg.C
C R : GAS CONSTANT FOR AIR (=8314.43/WMA), J/kg.K
C ROA,ROP : DENSITY OF AIR AND PARTICLE, kg/m^3
C HU : ABSOLUTE HUMIDITY OF AIR, kg MOISTURE/kg DRY AIR
C VAIR,VAXL : AIR VELOCITY THROUGH FAN INLET AND AXIAL VEL., m/s
C OMEGA : ANGULAR VELOCITY, RAD/s
C R1,R2,RP : RADIUS OF THROAT, CYCLONE AND PARTICLE, m
C XL : CALCULATED SETTLING DISTANCE OF A PARTICLE, cm
C N : NUMBER OF REVOLUTIONS PER SECOND
C FW,FH : FAN INLET WIDTH AND HEIGHT, m
C
OPEN(20,FILE='DATA.OUT',STATUS='NEW')

C
WMA = 28.9645
WMV = 18.051534
HC = 2500915.2
CPA = 1004.832
CPV = 1858.9392
CPL = 4186.8
R = 287.055724249
PI = 3.141592654

C
DO 20 II=1,100
WRITE(*,30)
READ(*,40) S1

```

```

IF((S1.EQ.'N').OR.(S1.EQ.'h')) THEN
  CLOSE(20,STATUS='KEEP')
  STOP
ENDIF
KK = 2
NEQ = 2
NORDER = 1
S = 'ADAMS PREDICTOR-CORRECTOR METHOD'
WRITE(*,50)
READ(*,*) D1,D2,FW,FH,DP,ROP
WRITE(*,60)
READ(*,*) P,T0,TW,AFLO
WRITE(*,70)
READ(*,*) DELT,FVALUE
C
  NN = FVALUE/DELT
  JJ = NN/100
  WRITE(*,80) JJ,NN,NN
  READ(*,*) INTV
C
  R1 = D1/100.0/2.0
  R2 = D2/100.0/2.0
  RP = DP*1.0D-06/2.0
  FW = FW/100.0
  FH = FH/100.0
  AFAN = FW*FH
  HU = HDBWB(T0,TW)
  ROA = ROTH(T0,HU)
  VAIR = AFLO/AFAN
  OMEGA = 2.0*VAIR/(R1 + R2)
  N = OMEGA/2.0/PI
  C1 = 4.5*VST(T0)/RP/RP/(ROP - ROA)
  C2 = OMEGA*OMEGA*R1**4
C
C  ARRAYS OF RADIUS OF REVOLUTION AND ITS DERIVATIVE WITH RESPECT
C  TO TIME (RADIAL VELOCITY) ARE INITIALIZED ...
C
  Y0(1,1) = R1
  Y0(2,1) = 0.0
  TIVALU = 0.0
C
  CALL START
C
C  NUMERICAL METHOD BEGINS BY CALLING SUBROUTINE 'RKPC'
C
  IK = 0
C
  WRITE(*,90) S
  WRITE(20,90) S
  WRITE(*,100) ((100.0*Y(I,1,M),M=1,NORDER),I=1,NEQ),TT
  WRITE(20,100)((100.0*Y(I,1,M),M=1,NORDER),I=1,NEQ),TT
C
  DO 10 J=2,NN+10

```

```

      IK = IK + 1
      CALL RKPC(KK,J,F)
      TT = TT + DELT
C
      IK1 = (IK/INTV)*INTV
      IF(IK1 .EQ. IK) THEN
        WRITE(*,100) ((100.0*Y(I,L,M),M=1,NORDER),I=1,NEQ),TT
        WRITE(20,100)((100.0*Y(I,L,M),M=1,NORDER),I=1,NEQ),TT
        IF((TT .LT. FVALUE) .AND. (Y(1,L,1) .LT. R2)) THEN
          PAUSE ''
          ELSEIF(TT .GE. FVALUE) THEN
            WRITE(*,110) TT
            PAUSE ' Going to MAIN MENU ...'
            GOTO 20
          ENDIF
        ENDIF
      ENDIF
C
      IF(Y(1,L,1) .GE. R2) THEN
        TT = TT*R2/Y(1,L,1)
        XL = N*TT*FH
        WRITE(*,120) TT,XL*100.0,DP,AFLO
        WRITE(20,120)TT,XL*100.0,DP,AFLO
        PAUSE ''
        GOTO 20
      ENDIF
      K = K + 1
10  CONTINUE
20  CONTINUE
C
C ***** FORMAT STATEMENTS *****
C
30  FORMAT(20(/)20X,39('-')//21X,'Press [Y] to CONTINUE and [N] ',
1    'to QUIT'/20X,39('-')//)
40  FORMAT(A1)
50  FORMAT(20(/)36X,'TYPE IN'//15X,49('-')/16X,'VALUES of D1, D2',
1    ' FW, FH, DP and ROP and <ENTER>'//22X,'D1 : THROAT ',
2    'diameter [cm]'/22X,'D2 : CYCLONE diameter [cm]'/
3    22X,'FW : FAN INLET width [cm]'/22X,'FH : FAN INLET ',
4    'height [cm]'/22X,'DP : PARTICLE diameter [microns]'/22X,
5    'ROP : PARTICLE density [kg/m^3]'/15X,49('-')//)
60  FORMAT(20(/)36X,'TYPE IN'//19X,41('-')/20X,'VALUES of P, T0',
1    ' TW and AFLO and <ENTER>'//22X,'P : ATMOSPHE',
2    'RIC pressure [Pa]'/22X,'T0 : DRY BULB temperature ',
3    '[C]'/22X,'TW : WET BULB temperature [C]'/22X,
4    'AFLO : AIRFLOW at fan inlet [m^3/s]'/16X,47('-')//)
70  FORMAT(20(/)36X,'TYPE IN'//20X,39('-')/21X,'VALUES of DELT ',
1    'and FVALUE and <ENTER>'//22X,'DELT : TIME ',
2    'step [s]'/22X,'FVALUE : TOTAL time of residence [s]'/
3    20X,39('-')//)
80  FORMAT(25(/)5X,71('-')/7X,'Give PRINTING INTERVAL [No. of ',
1    'steps to be skipped] and <ENTER>'//21X,'[It SHOULD be',
2    ' BETWEEN ',I5,' and ',I5,']'/22X,'[TOTAL number',
3    ' of DIVISIONS = ',I6,']'/5X,71('-')//)

```

```

90  FORMAT(/8X,'SOLUTIONS r [cm], dr/dt [cm/s] and corresponding',
1      ' STEP VALUES [s]/7X,66('-)//22X,A35/22X,35('-)//)
100  FORMAT(16X,3(1PD14.8,2X))
110  FORMAT(20(/)2X,76('-)//3X,'Residence time = ',1PD10.4,' s; NOT',
1      ' sufficient for the particle to SETTLE/2X,76('-)//)
120  FORMAT(4X,'Corrected residence time = ',1PD10.4,' s; Settling ',
1      ' LENGTH = ',1PD10.4,' cm/5X,'Particle diameter = ',
2      ' 1PD10.4,' microns and Air flow = ',1PD10.4,' m^3/s/2X,
3      ' 76('-)//)
      STOP
      END

```

```

C
C
C   THIS SUBROUTINE PROVIDES STARTING VALUES TO "RKPC"
C

```

```

$INCLUDE:'START.FOR'

```

```

C
C   THIS SUBROUTINE SOLVES A SET OF SIMULTANEOUS FIRST- AND
C   HIGHER-ORDER DIFFERENTIAL EQUATIONS BY 1. RUNGE-KUTTA,
C   2. ADAMS P-C AND HAMMING'S P-C METHODS [IN APPENDIX A.3.6.5]
C

```

```

$INCLUDE:'RKPC.FOR'

```

```

C
C   THIS FUNCTION DEFINES ALL THE FIRST ORDER DIFFERENTIAL
C   EQUATIONS ...
C

```

```

C
      DOUBLE PRECISION FUNCTION F(I, KK, M, T)
      COMMON /BLOK3/ Y(10,5,4), Z(10,5,4), Y0(10,4), YT(10,101), TT, ER1
      COMMON /BLOK5/ C1, C2
      DOUBLE PRECISION Y, Z, Y0, YT, TT, ER1, C1, C2, T

```

```

C
      IF(I .EQ. 1) THEN
        F = Y(2, KK, M)
      ELSEIF(I .EQ. 2) THEN
        F = -C1*Y(2, KK, M) + C2/(Y(1, KK, M))**3
      ENDIF

```

```

C
      RETURN
      END

```

```

C
C   HERE THE PSYCHROMETRIC PACKAGE "PSYCRO.FOR" IS INCLUDED ...
C   [LISTED IN APPENDIX A.5.1]
C

```

```

$INCLUDE:'PSYCRO.FOR'

```

A.5.9. Listing of program segment for the user to supply (example).

Program name : **INCLUDE.FOR**

```

C
C
C THIS PART OF THE PROGRAM IS TO BE SUPPLIED BY THE USER. THIS
C EXAMPLE IS GIVEN FOR PARBOILED ROUGH RICE (Ref. Ph.D. THESIS
C of Prof. Prabir K. Chandra, 1983).
C
C
C THIS SUBROUTINE CALCULATES dM/dX FROM PAGE'S THIN-LAYER
C EQUATION
C
C
C SUBROUTINE THINLR(Y,Y0,GFLO,RH,XMG,XME,DELT,TA,L)
C DOUBLE PRECISION Y(10,5,4),Y0(10,4),GFLO,RH,XMG,XME,DELT,TA
C DOUBLE PRECISION A,B,C,D,A1,B1,C1,D1,P,Q,XMR,TI,ZAA,RH1
C
C RH1 = RH/100.0
C XMR = (Y(1,L,1) - XME)/(XMG - XME)
C IF(Y(4,L,1) .GT. 99.99999) THEN
C   A = -12.382707
C   B = 0.304496
C   C = 1.882463
C   D = 1.173742
C   A1 = -0.251444
C   B1 = 2.225801
C   C1 = 2.719537D-03
C   D1 = 0.99337
C
C   P = DEXP(A + B*RH1 + C*DLOG(TA*1.8 + 32.0)+ D*XMG)
C   Q = A1 + B1*RH1 + C1*TA + D1*XMG
C ELSE
C   A = -0.1700425
C   B = -2.173133D-02
C   C = 3.145021D-03
C   D = 0.149104
C   A1 = 0.589744
C   B1 = 0.190314
C   C1 = 1.2106D-04
C   D1 = 7.833534D-02
C
C   P = A + B*RH1 + C*TA + D*XMG
C   Q = A1 + B1*RH1 + C1*TA + D1*XMG
C ENDIF
C
C IF(XMR .LE. 0.0) XMR = (-1)*XMR
C IF(Y(2,L,1) .LT. 1.0D-15) THEN
C   XMR = Y(1,L,1)/XMG
C   XME = 0.0

```

```

ENDIF
IF(RH1 .GE. 1.0) THEN
  RH1 = 0.999999999
  XMR = 1.0D-07
  XME = Y(1,L,1)
ENDIF
C
ZAA = -(XMG - XME)/(GFLO*60.0)
TI = (-DLOG(XMR)/P)**(1.0/Q) + DELT/60.0
IF(TI .EQ. 0.0) THEN
  Y0(5,1) = 0.0
  RETURN
ELSE
  Y0(5,1) = ZAA*P*Q*(TI**(Q - 1.0))*DEXP(-P*(TI**Q))
ENDIF
C
RETURN
END
C
C
C FDIFF(T) EXPRESSES MASS DIFFUSIVITY OF ROUGH RICE IN m^2/s
C
C
DOUBLE PRECISION FUNCTION FDIFF(T)
DOUBLE PRECISION T
C
FDIFF = 4.157444444D-06*DEXP(-3748.6/(T+273.15))
RETURN
END
C
C
C FHM(H,RO,CP) COMPUTES MASS TRANSFER COEFFICIENT IN m/s
C
C
DOUBLE PRECISION FUNCTION FHM(H,RO,CP)
DOUBLE PRECISION H,RO,CP
C
FHM = H/RO/CP
RETURN
END
C
C
C FHFG(TG,XMG) COMPUTES LATENT HEAT OF EVAPORATION FROM GRAIN
C SURFACE IN J/kg
C
C
DOUBLE PRECISION FUNCTION FHFG(TG,XMG)
DOUBLE PRECISION TG,XMG
C
FHFG = 1000.0*(1862.453 - 1.7575*TG)*XMG**(-0.21304)
RETURN
END
C

```



```

C
C FXME(T,RH) COMPUTES EQUILIBRIUM MOISTURE CONTENT IN
C FRACTION db
C
C
C DOUBLE PRECISION FUNCTION FXME(T,RH)
C DOUBLE PRECISION T,RH
C
C FXME = (4.51 + 0.069*RH + 8.837*DSQRT(RH) - 0.027*DSQRT(RH)*
1 (T + 273.15))/100.0
C RETURN
C END

```

```

C
C
C FH(GPA) COMPUTES HEAT TRANSFER COEFFICIENT IN W/m^2.C
C
C
C DOUBLE PRECISION FUNCTION FH(GPA)
C DOUBLE PRECISION GPA
C
C FH = 24.19*GPA**(0.37)
C RETURN
C END

```

A.5.10. Complete program listing for the concurrent drying simulation using (i) diffusion and (ii) thin layer approaches.

Program name : **DRY2000.FOR**

```

C
C
C THIS PROGRAM COUPLED WITH THE SUBROUTINES 'GEOMTY', 'COEFF',
C 'TRIDGL', 'START' AND 'RKPC' SIMULATES DRYING PROCESS
C
C Program Developed by: Prof. Prabir K. Chandra
C
C
C DIMENSION AM(2),TM(101),X(101)
C COMMON /BLOK1/ TIME,TTIME,DELT,DD,DD1,XME,T0
C COMMON /BLOK2/ A(101),B(101),D(101),R(101),V(101),XM(1,101),
1 XMS(101),AR(101),V0,R1,DELR,N
C COMMON /BLOK3/ Y(10,5,4),Z(10,5,4),Y0(10,4),YT(10,101),XX,ER1
C COMMON /BLOK4/ XMF,XIVALU,DELX,NEQ,NORDER,K,L
C COMMON /PROP/ ROA,ROG,DIFF,CPP,HFG,PORSTY,H,HM,ARATIO,GPA,GPP,PI
C COMMON /PSYCR/ P,RG,WMA,WMV,HC,CPA,CPV,CPL
C EXTERNAL F
C DOUBLE PRECISION
C TIME,TTIME,DELT,DD,DD1,XME,T0,XM0,XMG,GFLO,WMA
C DOUBLE PRECISION A,B,D,R,V,XM,XMS,AR,AM,TM,X,V0,R1,DELR,TA,TG,TW

```

DOUBLE PRECISION Y,Z,Y0,YT,XX,ER1,XMF,XIVALU,XMEAN,ARATIO,P,HC
 DOUBLE PRECISION ROA,ROG,DIFF,CPA,CPP,CPV,CPL,PORSTY,H,HM,PI,W MV
 DOUBLE PRECISION GPA,GPP,HFG,VH,XBED,RH,DELX,AVEL,GVEL,AREA,RG
 DOUBLE PRECISION F,PST,FDIFF,VTH,ROTH,FHFG,FHM,FXME,PVH,RHHT,FH
 DOUBLE PRECISION HDBWB,HPV
 CHARACTER S *15, S2 *46, S3 *35
 CHARACTER S1 *20, S0 *1, S4 *19

C
 C AM, TM and X : TEMPORARY STORAGE FOR dM/dX, PRINTING ARRAYS
 C for TIME and DEPTH, RESPECTIVELY.
 C DELR, DELT, TIME,TTIME : STEP SIZE IN SPACE (m) AND TIME (s),
 C VARIABLE AND TOTAL TIME OF EXPOSURE (s), RESPECTIVELY.
 C D, A, B and N: DIAGONAL, UPPER, LOWER ELEMENTS and ROW OR
 C COLUMN SIZE OF TRIANGULAR MATRIX.
 C Y,Y0 : ARRAYS OF GRAIN MOISTURE [Y(1,L,1)], AIR ABSOLUTE
 C HUMIDITY [Y(2,L,1)], AIR TEMPERATURE [Y(3,L,1)], GRAIN
 C TEMPERATURE [Y(4,L,1)] and ARRAYS OF INITIAL CONDITIONS.
 C R, XM: ELEMENTS OF CONSTANT VECTOR AND SOLUTION VECTOR (MC).
 C ROA, DIFF, CPA, HM and H : DENSITY (kg/m^3), DIFFUSIVITY
 C (m^2/s), SPECIFIC HEAT (J/kg.C), MASS AND HEAT TRANSFER
 C COEFFICIENTS (m/s) and ($\text{W/m}^2.\text{C}$), RESPECTIVELY.
 C AR(J), V(J) : J-TH AREA PERPENDICULAR TO FLOW (m^2) AND
 C SEGMENT VOLUME (m^3), RESPECTIVELY.
 C TA,T0,XM0 : TEMPERATURE OF HEATED AIR, AMBIENT TEMPERATURE
 C OF AIR (C), AND INITIAL ABSOLUTE HUMIDITY OF AIR.
 C TW : WET BULB TEMPERATURE [C].
 C TG,XMG,XMF : INITIAL GRAIN TEMPERATURE, INITIAL MOISTURE
 C OF GRAIN AND FINAL MOISTURE DESIRED (% db).
 C AVEL,GVEL : AIR VELOCITY [m^3/S], GRAIN VELOCITY [kg/s].
 C AREA,XBED,NN : EFFECTIVE DRYER AREA [m^2], BED DEPTH [m] and
 C NUMBER OF IMAGINARY GRAIN LAYERS ACROSS THE BED.
 C CPP,ROG,PORSTY : SPECIFIC HEAT [J/kg.C], BULK DENSITY
 C [kg/m^3] and POROSITY OF GRAIN/PRODUCT.
 C GPA,GPP : DRY AIR AND GRAIN FLOW RATE IN [$\text{kg/m}^2.\text{s}$].
 C HFG,VH,RH : LATENT HEAT OF EVAPORATION FROM GRAIN SURFACE
 C [J/kg], SPECIFIC VOLUME [$\text{m}^3 \text{ moist/kg dry}$], RELATIVE
 C HUMIDITY [%] OF AIR.
 C R1,RG : PRODUCT RADIUS [m] and GAS CONSTANT for AIR [J/kg.K].
 C WMA,W MV : MOLECULAR WEIGHT OF AIR AND WATER.
 C P,HC: ATMOSPHERIC PRESSURE [Pa] and HEAT CAPACITY OF AIR [J/kg]
 C CPA,CPV,CPL: SPECIFIC HEAT OF AIR, VAPOR, LIQUID WATER [J/kg.C]
 C ARATIO: RATIO BETWEEN HEAT TRANSFER AREA AND BED VOLUME
 C [m^2/m^3].

OPEN(20,FILE='DATA.OUT',STATUS='NEW')

C
 C SPECIFIC HEAT OF AIR, WATER VAPOR AND LIQUID WATER ...
 C

CPA = 1004.832
 CPV = 1858.9392
 CPL = 4186.8
 CPP = 1674.7

C

```

RG = 287.055724249
WMA = 28.9645
WMV = 18.051534
HC = 2500915.2

```

C

```

PI = 3.141592654
S = 'IMPLICIT SCHEME'
DO 80 II=1,100
  WRITE(*,90)
  READ(*,100) S0
  IF((S0.EQ. 'N') .OR. (S0.EQ. 'n')) THEN
    CLOSE(20,STATUS='KEEP')
    STOP
  ENDIF
  WRITE(*,110)
  READ(*,*) KL
  IF(KL.EQ. 1) THEN
    S4 = 'DIFFUSION EQUATION'
  ELSE
    S4 = 'THIN-LAYER EQUATION'
  ENDIF
  WRITE(*,120) S
  READ(*,*) LX
  IF(LX.EQ. 2) THEN
    S1 = 'CYLINDRICAL GEOMETRY'
  ELSEIF(LX.EQ. 3) THEN
    S1 = 'SPHERICAL GEOMETRY'
  ELSE
    GOTO 80
  ENDIF
  S2 = 'SIDE 1 : CONVECTIVE BC; CENTER : DERIVATIVE BC'
  WRITE(*,130)
  READ(*,*) KK
  IF(KK.EQ. 1) THEN
    S3 = 'FOURTH-ORDER RUNGE-KUTTA METHOD'
  ELSEIF(KK.EQ. 2) THEN
    S3 = 'ADAMS PREDICTOR-CORRECTOR METHOD'
  ELSEIF(KK.EQ. 3) THEN
    S3 = 'HAMMING'S PREDICTOR-CORRECTOR METHOD'
  ENDIF
  WRITE(*,140)
  READ(*,*) P,T0,TW,TA,TG,XMG,XMF
  WRITE(*,150)
  READ(*,*) AVEL,GVEL,AREA,XBED,NN
  WRITE(*,160)
  READ(*,*) CPP,ROG,PORSTY

```

C

```

XBED = XBED/100.0
XM0 = HDBWB(T0,TW)
XMG = XMG/100.0
XMF = XMF/100.0
DELX = XBED/NN
NEQ = 4

```

```

NORDER = 1
XIVALU = 0.0
C
C ARRAYS OF GRAIN MOISTURE [Y(1,1,1)], AIR MOISTURE/HUMIDITY AND
C TEMPERATURE [Y(2,1,1) AND Y(3,1,1)] AND GRAIN TEMPERATURE
C [Y(4,1,1)] ARE INITIALIZED ...
C
      Y0(1,1) = XMG
      Y0(2,1) = XM0
      Y0(3,1) = TA
      Y0(4,1) = TG
C
      TA = (TA + TG)/2.0
      VH = VTH(TA,XM0)
      GPA = AVEL/VH/(AREA*(1.0 - PORSTY))
      H = FH(GPA)
C
      CALL START
C
      IF(KL.EQ. 1) THEN
        WRITE(*,170)
        READ(*,*) R1,N
        X1 = X1/1000.0
        DELR = R1/N
C
        CALL GEOMTY(LX)
C
        DIFF = FDIFF(TA)
C
      ELSE
        WRITE(*,180)
        READ(*,*) R1
        X1 = X1/1000.0
      ENDIF
      IF(LX.EQ. 2) THEN
        ARATIO = 2.0*(1.0 - PORSTY)/R1
      ELSEIF(LX.EQ. 3) THEN
        ARATIO = 3.0*(1.0 - PORSTY)/R1
      ENDIF
      GFLO = GVEL/ROG/AREA
      DELT = DELX/GFLO
C
C HM : MASS TRANSFER COEFFICIENT, m/s, DIFF : DIFFUSIVITY, m^2/s
C
      HFG = FHFG(TG,XMG)
      ROA = ROTH(TA,XM0)
      HM = FHM(H,ROA,CPA)
      GPP = GVEL/(1.0 + XMG)/AREA
C
      RH = 100.0*RHHT(Y(2,1,1),TA)
      XME = FXME(TA,RH)
C
      JJ = NN/100 + 1

```

```

WRITE(*,190) JJ,NN,NN
READ(*,*) INTV
IF(KL.EQ. 1) THEN
C
C  INITIAL CONDITION IS ATTRIBUTED ...
C
      DO 10 J=1,N+1
        XM(1,J) = XMG
10    CONTINUE
C
      DD = -DELX*DELX/DIFF/DELT
      DD1 = DIFF*(AR(1) + AR(2))/(2.0*AR(1)*HM*DELX)
      WRITE(20,200) S3,S,S1,S2
      WRITE(*,210) S3,S,S1,S2
      XM(1,1) = (XME + DD1*XM(1,2))/(1.0 + DD1)
      XM(1,N+1) = XM(1,N)
      ENDIF
C
C  TIME IS INITIALIZED ... INITIAL CONDITION IS ATTRIBUTED
C
      TIME = 0.0
      TM(1) = TIME
      AM(1) = XMG
      IK = 0
      LL = 1
      L = 1
      DO 50 LJ = 2,NN+1
C
C  TIME IS INCREMENTED BY DELT ...
C
        TIME = TIME + DELT
        XX = XX + DELX
        IK = IK + 1
C
        IF(KL.EQ. 1) THEN
          CALL COEFF
          CALL TRIDGL(N)
C
          XMS(1) = (XME + DD1*XMS(2))/(1.0 + DD1)
          XMS(N+1) = XMS(N)
C
          XMEAN = 0.0
          DO 20 J=1,N
            XMEAN = XMEAN + V(J)*(XMS(J) + XMS(J+1))/2.0/V0
20        CONTINUE
          AM(2) = XMEAN
C
C  Y0(5,1) is dM/dX and Y0(6,1) is d(XM0)/dX ...
C
          Y0(5,1) = (AM(2) - AM(1))/DELX
          ELSE
            CALL THINLR(Y,Y0,GFLO,RH,XMG,XME,DELT,TA,L)
          ENDIF

```

```

        Y0(6,1) = -(GPP/GPA)*Y0(5,1)
C
        CALL RKPC(KK,LJ,F)
C
        TA = (Y(3,L,1) + Y(4,L,1))/2.0
        RH = 100.0*RHHT(Y(2,L,1),TA)
        IF(RH .GE. 100.0) THEN
            WRITE(*,220) RH
            PAUSE ''
            GOTO 60
        ENDIF
        ROA = ROTH(TA,Y(2,L,1))
        IF(KL .EQ. 1) THEN
            HM = FHM(H,ROA,CPA)
            DIFF = FDIFF(TA)
            DD1 = DIFF*(AR(1) + AR(2))/(2.0*AR(1)*HM*DELR)
        ENDIF
        HFG = FHFG(Y(4,L,1),Y(1,L,1))
        XME = FXME(TA,RH)
        IK1 = (IK/INTV)*INTV
        IF(IK1 .EQ. IK) THEN
            LL = LL + 1
            TM(LL) = TIME
            X(LL) = XX
            DO 30 I=1,NEQ
                YT(I,LL) = Y(I,L,1)
30        CONTINUE
        ENDIF
        IF(Y(1,L,1) .LE. XMF) GOTO 60
        IF(KL .EQ. 1) THEN
C
C      OLD VALUES ARE GIVEN NEW VALUES FOR THE NEXT TIMESTEP ...
C
            DO 40 J=1,N+1
                XM(1,J) = XMS(J)
40        CONTINUE
            ENDIF
            AM(1) = AM(2)
50        CONTINUE
C
60        WRITE(*,230) S4
        WRITE(20,240) S4
        DO 70 JK=1,LL
            X(JK) = 100.0*X(JK)
            YT(1,JK) = 100.0*YT(1,JK)
            WRITE(*,250) X(JK),YT(1,JK),(YT(I,JK),I=2,NEQ),TM(JK)
            WRITE(20,250)X(JK),YT(1,JK),(YT(I,JK),I=2,NEQ),TM(JK)
70        CONTINUE
            WRITE(*,260)
            WRITE(20,260)
            PAUSE 'Going to the MAIN MENU ...'
80        CONTINUE
C

```

C ***** FORMAT STATEMENTS *****

C

```

90  FORMAT(20(/)20X,39('-')//21X,'Press [Y] to CONTINUE and [N] ',
1    'to QUIT'/20X,39('-')////)
100 FORMAT(A1)
110 FORMAT(20(/)14X,49('-')/24X,'Press [1] for DIFFUSION equation/'
1    16X,'Press [2] for THIN-LAYER equation - and <ENTER>/'
2    14X,49('-')////)
120 FORMAT(20(/)1X,79('#')/1X,'#',77X,'#'/1X,'#',27X,'GRAIN ',
1    'DRYING SIMULATION',27X,'#'/1X,'#',27X,23('-'),27X,'#'
1    /1X,'#',77X,'#'/1X,'#',31X,A15,31X,'#'/1X,'#',77X,'#'
1    1X,'#',24X,'DECLARE PRODUCT CONFIGURATION',24X,'#'/1X,
1    '#',24X,29('-'),24X,'#'/1X,'#',77X,'#'/1X,'#',31X,
1    '2: CYLINDRICAL',32X,'#'/1X,'#',31X,'3: SPHERICAL',34X,
1    '#'/1X,'#',77X,'#'/1X,'#',3X,'[Type any OTHER number',
1    ' (EXCEPT 1, 2 and 3) to RETURN to the BEGINNING]',4X,
1    '#'/1X,'#',77X,'#'/1X,79('#')////)
130 FORMAT(20(/)1X,79('#')/1X,'#',77X,'#'/1X,'#',27X,'GRAIN ',
1    'DRYING SIMULATION',27X,'#'/1X,'#',77X,'#'/1X,'#',30X,
1    'MAIN OPTION MENU',31X,'#'/1X,'#',77X,'#'/1X,'#',25X,
1    'CHOOSE ANY METHOD BY NUMBER',25X,'#'/1X,'#',25X,
1    27('-'),25X,'#'/1X,'#',77X,'#'/1X,'#',30X,'1: RUNGE',
1    'KUTTA',33X,'#'/1X,'#',30X,'2: ADAMS P-C',35X,'#'/1X,
1    '#',30X,'3: HAMMINGS P-C',32X,'#'/1X,'#',77X,'#'/1X,'#',
1    23X,'[Type any OTHER number to QUIT]',23X,'#'/1X,'#',77X,
1    '#'/1X,79('#')////)
140 FORMAT(20(/)10X,59('-')/12X,'Give VALUES of P, T0, TW, TA, TG,',
1    ' XMG and XMF and <ENTER>'//20X,'P   : ATMOSPHERIC ',
2    'PRESSURE [Pa]'/20X,'T0   : AMBIENT TEMPERATURE [C]'/20X,
3    'TW   : WET BULB TEMPERATURE [C]'/20X,'TA   : HEATED AIR',
4    'TEMPERATURE [C]'/20X,'TG   : INITIAL GRAIN TEMPERATURE',
5    '[C]'/20X,'XMG : INITIAL GRAIN MOISTURE (% db)'/20X,
6    'XMF : FINAL MOISTURE DESIRED (% db)'/10X,59('-')////)
150 FORMAT(20(/)10X,58('-')/11X,'Give VALUES of AVEL, GVEL, AREA,',
1    ' XBED and NN and <ENTER>'//20X,'AVEL : AIR FLOW RATE ',
2    '[m^3/s]'/20X,'GVEL : GRAIN FLOW RATE [kg/s]'/20X,'AREA',
3    ' : DRYER EFFECTIVE AREA [m^2]'/20X,'XBED : GRAIN BED ',
4    'DEPTH [cm]'/20X,'NN   : NUMBER OF IMAGINARY GRAIN
LAYERS'
5    /10X,58('-')////)
160 FORMAT(20(/)16X,48('-')/18X,'Give VALUES of CPP, ROG and ',
1    'PORSTY and <ENTER>'//20X,'CPP   : SPECIFIC HEAT of ',
1    'GRAIN [J/kg.C]'/20X,'ROG   : BULK DENSITY of GRAIN ',
1    '[kg/m^3]'/20X,'PORSTY : POROSITY of GRAIN [fraction]'/
1    16X,48('-')////)
170 FORMAT(20(/)4X,72('-')/5X,'Give RADIUS in [mm], and No. of ',
1    'DIVISIONS along the radius and <ENTER>'//18X,'[Number ',
2    'of DIVISIONS should NOT EXCEED 100]'/4X,72('-')////)
180 FORMAT(20(/)22X,36('-')/23X,'Give the RADIUS in [mm] and <ENTER>'
1    /22X,36('-')////)
190 FORMAT(20(/)5X,71('-')/7X,'Give PRINTING INTERVAL [No. of ',
1    'steps to be skipped] and <ENTER>'//21X,'[It SHOULD be',
1    ' BETWEEN ',15,' and ',15,']'/22X,'[TOTAL number',

```

```

1          ' of DIVISIONS = 'I5,']/5X,71('-')////)
200 FORMAT(/22X,['A35,']/21X,A15,' - ',A20/21X,38('-')/16X,['
1          A46,']/)
210 FORMAT(20(/)22X,['A35,']/21X,A15,' - ',A20/21X,38('-')/16X,
1          '['A46,']/)
220 FORMAT(20(/)2X,76('-')/3X,'THE AIR IS COMPLETELY SATURATED ',
1          '[RH = ',F8.4,'%] - CAN NOT DRY ANY MORE ...'/2X,76('-')
2          ////)
230 FORMAT(20(/)4X,73('-')/17X,A19,' used for MOISTURE GRADIENT'/
1          17X,46('-')/6X,'DEPTH',6X,'MOISTURE',5X,'HUMIDITY',
2          5X,'AIR TEMP.',3X,'GRAIN TEMP.',4X,'TIME'/6X,['cm'],8X,
3          '['% db]',5X,['fraction'],7X,['C'],10X,['C'],9X,['s']/4X,
4          73('-'))
240 FORMAT(/4X,73('-')/17X,A19,' used for MOISTURE GRADIENT'/17X,
1          46('-')/6X,'DEPTH',6X,'MOISTURE',5X,'HUMIDITY',
2          5X,'AIR TEMP.',3X,'GRAIN TEMP.',4X,'TIME'/6X,['cm'],8X,
3          '['% db]',5X,['fraction'],7X,['C'],10X,['C'],9X,['s']/4X,
4          73('-'))
250 FORMAT((2(F11.2,2X),F11.5,2X,3(F11.2,2X)))
260 FORMAT(4X,73('-')////)
      STOP
      END

```

C

C

C THIS SUBROUTINE CALCULATES AREA AND VOLUME OF DIFFERENT
C SHAPES [LISTED IN APPENDIX A.3.7.2]

C

C

\$INCLUDE:'GEOMTY.FOR'

C

C

C THIS SUBROUTINE CALCULATES THE COEFFICIENTS FOR SUBROUTINE TRIDGL...
C [LISTED IN APPENDIX A.5.6]

C

C

\$INCLUDE:'COEFF.FOR'

C

C

C 'TRIDGL.FOR' SOLVES A 1-D PARABOLIC PDE USING TRIDIAGONAL
C MATRIX. [LISTED IN APPENDIX A.3.7.3]

C

C

\$INCLUDE:'TRIDGL.FOR'

C

C

C THE SUBROUTINE "START" PROVIDES STARTING VALUES TO "RKPC"
C [LISTED IN APPENDIX A.3.7.4]

C

C

\$INCLUDE:'START.FOR'

C

C

C THE SUBROUTINE "RKPC" SOLVES A SET OF SIMULTANEOUS FIRST-


```

C   AND HIGHER-ORDER DIFFERENTIAL EQUATIONS.
C   [LISTED IN APPENDIX A.3.6.5]
C
C
C$INCLUDE:'RKPC.FOR'
C
C
C   THIS PART "INCLUDE.FOR" SHOULD BE WRITTEN BY THE USER
C   ACCORDING TO HIS OR HER SPECIFIC NEEDS AND CONDITIONS ...
C   [LISTED IN APPENDIX A.5.8]
C
C$INCLUDE:'INCLUDE.FOR'
C
C
C   FUNCTION TO DEFINE ALL THE FIRST-ORDER ODEs ...
C
C
C   DOUBLE PRECISION FUNCTION F(I, KK, M, T)
C   COMMON /BLOK3/ Y(10,5,4), Z(10,5,4), Y0(10,4), YT(10,101), TT, ER1
C   COMMON /PROP/ ROA, ROG, DIFF, CPP, HFG, PORSTY, H, HM, ARATIO, GPA, GPP, PI
C   COMMON /PSYCR/ P, R, WMA, WMV, HC, CPA, CPV, CPL
C   DOUBLE PRECISION Y, Z, Y0, YT, TT, ER1, T, A1, A2, A3, A4, DT
C   DOUBLE PRECISION ROA, ROG, DIFF, CPA, CPP, CPV, CPL, HFG, PORSTY, H, HM
C   DOUBLE PRECISION ARATIO, GPA, GPP, R, P, HC, WMA, WMV, PI
C
C   DT = Y(3, KK, 1) - Y(4, KK, 1)
C   IF(I .EQ. 1) THEN
C     F = Y0(5, 1)
C   ELSEIF(I .EQ. 2) THEN
C     F = -(GPP/GPA)*Y0(5, 1)
C   ELSEIF(I .EQ. 3) THEN
C     A1 = H*ARATIO*DT
C     A2 = GPA*(CPA+CPV*Y(2, KK, 1))
C     F = -A1/A2
C   ELSEIF(I .EQ. 4) THEN
C     A1 = H*ARATIO*DT
C     A2 = GPP*(CPP+CPL*Y(1, KK, 1))
C     A3 = A1/A2
C     A4 = (HFG+CPV*DT)/A2
C     F = A3 - A4*GPA*Y0(6, 1)
C   ENDIF
C
C   RETURN
C   END
C
C
C   "PSYCRO.FOR" IS INCLUDED TO TAKE CARE OF ANY PSYCHROMETRIC RELATIONS
C   [LISTED IN APPENDIX A.5.1]
C
C
C$INCLUDE:'PSYCRO.FOR'
C

```

Index

—A—

Absolute error, 198–200
Absolute humidity, 291–295; 297;
308; 310
Absolute viscosity, 1–2
Adams-Bashforth method, 179–
180
Adams-Moulton method, 179;
181
Addition, 16; 126–127
ADI method, 229; 234
Algebraic equation, 125; 133;
141; 143; 153
See also: 351–358
Analytic (equation), 170–171;
173; 179; 196
Analytical solution, 156; 163;
173; 290; 304
Approximate solution, 251
Arithmetic expression, 14; 16–17;
53; 62
Arithmetic IF, 53
Arithmetic statement, 16; 62; 64;
69; 73–74
Array, 32–34; 53–55; 60–61
Asymptotic regression, 163
Augmented matrix, 135–136;
138–139; 142

—B—

Back substitution, 136; 138; 141;
152
Backward difference, 80; 84;
179–180
Banded matrix, 127; 146; 151–
152
Biot number, 281
Bisection, 90; 93; 95–99; 101
BLOCK COMMON, 74
BLOCK DATA, 65

Block IF, 26
Boundary condition, 196–197;
199
Boundary value problems, 169–
170; 186; 195; 197

—C—

Cauchy condition, 206; 224
Central difference, 80–87
Centrifugal force, 306
Characters, 15; 36; 41–44; 62; 69
Chopping off, 240
Closed formula, 178–183
Coefficient matrix, 126; 139–141;
143–145; 151; 153
Cofactor, 132
Column vector, 126; 135–136;
141; 144
COMMON, 57; 60–62; 65; 69–
70; 72–74; 76
Concurrent flow, 310; 316
Conductivity, 302; 304–305
Constant vector, 126
Convective boundary, 207; 210;
219
Convergence, 92; 94–95; 98; 100;
111; 114–116; 121; 123; 146–
147; 149; 151; 178; 184; 196–
197; 199; 200
Conversion factor, 1–2; 264
Corrector, 174; 178; 181; 183–
187; 192; 195; 198
Correlation coefficient, 167
Crank-Nicolson method, 216;
219–224
Cyclone, 290; 306
See also: 479–483

—D—

DATA, 15; 22; 35; 46–47; 49; 53;
66; 69; 74
Definite integral, 105–106
Density, 2; 9; 292–293; 302; 304–
305; 307–308; 313; 318

Dependent variable, 157–158;
162
Derivative boundary, 207–208;
209; 213–214; 223; 313
Determinant, 131
Dew point, 295
Diagonal elements, 127; 140;
153–154; 222
Differential equation, 169–171;
178; 185; 186; 193–195; 197
Differentiation, 77; 105–106; 109
Diffusion, 290; 311–313; 316;
318–319
Diffusivity, 304; 313
DIMENSION, 32–33; 37; 39; 49;
51; 53; 55; 61; 65; 68; 70
Dimensional analysis, 263–264;
266; 269; 275; 282; 284; 286
Dimensionless group/number,
264–265; 269–270; 274–278;
280–282; 284; 286
Dirichlet condition, 206; 209;
223; 231
Discretization, 244
Division, 16
DO WHILE, 50; 63
DO-loop, 28–31; 33–35; 38; 49;
63; 75
DOUBLE PRECISION, 15; 85
Dry bulb, 290–292; 318
Drying, 157; 290; 309; 311–312;
318–319
See also: 484–486
Dynamic similarity, 282; 283–285

—E—

Element equations, 244–245
Element stiffness matrix, 256
Elements, 244
Embedding elements, 252
Elliptic, 201; 235
End correction, 108; 110; 122–
123
END, 19
Energy balance, 310

Energy, 1–2
Enthalpy, 290–291; 293
ENTRY, 66
EQUIVALENCE, 54–55; 76
Error estimation, 240
Euler method, 172–173; 186
Euler number, 278; 281; 284
Euler-Cauchy method, 172
Explicit scheme, 203; 212; 219;
229
Explosion, 303
Exponentiation, 16
External effects, 256
External file, 20; 36
Extrapolation, 82; 85; 109; 112

—F—

False Position method, 90; 93; 95;
97–98; 101
Fanning friction factor, 272
Fick's first and second laws, 312–
313
Finite difference grid, 204
Finite difference, 79; 179; 240
See also: 321–328
Finite elements, 243
Finite element methods, 243
Finned tube heat exchanger, 289;
303
See also: 467–472
First-order methods, 80; 83; 171–
172; 189; 192; 197
FLOAT, 49; 74
Floating point, 14; 17
Flow chart, 29
Flow diagram, 96; 99; 114; 138;
142; 148
Flow symbols, 29
Force, 1–4; 264; 267; 271; 279;
282; 283–285
FORMAT, 15; 20; 40; 52; 75
Forward difference, 80; 83
Fourier number, 281
Fourier's law, 202; 209

Fourth-order methods, 82; 108;
 110; 176; 179; 182; 186; 193
 FPS, 1–2; 4; 7
 Free format, 19; 38; 63
 Froude number, 281; 284
 Functional equation, 265; 266;
 276; 279
 Functions, 28; 49; 52; 55; 62; 69;
 74

—G—

Galerkin method, 253
 Gauss elimination method, 135;
 138; 141–143; 146–147; 151–
 152; 219; 229
 Gauss-Jordan elimination, 141–
 143; 153; 160; 161
 Gauss-Legendre method, 106;
 116; 120; 122
 Gauss-Seidel iteration/iterative
 method, 146; 229
 Geometric ratio, 284
 Geometric similarity, 282
 Global coordinates, 258
 Global numbering system, 257
 GO TO, 22; 40
 Graetz number, 281
 Grashof number, 281

—H—

Hamming's method, 182–183;
 186; 192; 195; 198
 Heat conduction, 202; 228
 Heat exchanger, 289; 299; 303
 Heat source/sink, 299
 Heat transfer coefficient, 6; 264;
 270; 290; 300; 303; 310
 Heat transfer, 266; 270; 281
 Higher-order methods, 80; 81–82;
 86–87; 186
 Homogeneous equation, 170
 Humid heat, 290–291; 294
 Humid volume, 293
 Hyperbolic, 202

—I—

Identity/unit matrix, 127; 130;
 141; 143–144
 IFIX, 49
 Imaginary node point, 208; 225
 Imaginary temperature, 208; 231
 Implicit scheme, 212; 216; 219;
 229; 234; 289; 294; 302; 314
 Implicit, 5; 7
 Implied DO, 38
 Improper integral, 105–106; 119
 Improved Euler method, 173; 186
 INCLUDE, 67
 Independent equation, 126
 Independent variable, 157–158;
 160; 162; 166
 Inflection point, 92
 Inherent error, 238
 Initial condition, 170; 184; 186;
 189; 196–197; 308
 Initial value problems, 170
 Integers, 14; 28; 62; 74
 Integrand, 105–106; 109; 116;
 119
 Integration, 83; 85; 105
See also: 341–350
 International system, 1
 Interpolation function, 244; 247
 Intrinsic, 19; 49; 52
 Inverse of a matrix, 130
 Isometric properties, 202
 Iterative method, 90
 Iterative-convergence, 289

—K—

Keenan's equation, 292
 KEEP, 36
 Kinematic similarity, 283
 Kinematic viscosity, 1–2

—L—

Laplace's equation, 202

Latent heat, 290–291; 293; 296;
310

Least square, 157

Least-square criterion, 158; 160

Length, 1; 264; 267; 277; 279;
282; 284

Lewis number, 281

Library functions, 49; 56; 62; 74

Line function, 62

Linear algebraic equations, 125;
133; 141; 143; 153

Linear regression, 157

Linearization, 163

Local minimum, 92

Logical IF, 23

Logical, 11; 15; 23; 51

Logistic regression, 164

Lower triangular matrix, 127

—M—

Mach number, 281

Mass balance, 310

Mass diffusion, 202

Mass, 1–4, 263; 267; 272; 275;
277; 279; 283

Mathematical modeling, 263;
282; 306

Matrix inversion, 130; 141

Matrix, 32; 126–146; 151–154;
159–161; 164–166; 245; 247;
256; 258

Mean-Value theorem, 182

Metric system, 1

MKS system, 1; 3; 291

Model testing, 282

Modified Newton, 90; 93–94; 98;
100; 101

Modifier, 183–184

Multilinear regression, 157; 164

Multiple root, 90; 92; 97; 101

Multiplication (matrix), 126–128;
136; 144

Multiplication, 16

Multiplicity, 90

—N—

Nested DO, 35

Neumann condition, 207; 225;
304; 314

NEW, 36

Newton-Raphson, 90; 98; 100–
101

Nodes, 244–248

Nonhomogeneous equation, 170

Nonlinear regression, 157; 163

Normal equations, 159; 165

Nusselt number, 281

—O—

OLD, 36

One-dimensional, 229; 234; 249;
252

Open formula, 179–180; 182

OPEN, 36; 66; 68; 74

Ordinary differential equation
(ODE), 169; 171; 178; 195;
201; 308

See also: 377–408

Orthogonal, 116

Orthogonality, 131

Output, 15; 19; 36; 42; 45; 57; 66

Overflow, 239

—P—

Page's equation (regression), 162

Panels, 108; 115; 120

Parabolic PDE, 202; 228; 235;
311

PARAMETER, 33

Partial differential equation
(PDE), 169; 201; 206; 228,
289; 311

See also: 409–446

PAUSE, 67

Peclet number, 281

Physical quantities, 263–264

Pi theorem, 275; 277

Pivot element, 136; 139; 143;
145; 154
Pivotal condensation, 139
Poisson's equation, 202
Polynomial regression, 157; 160;
165
Polynomial, 89; 100; 245
Polynomials, 109; 116; 122
Power, 1–2
Prandtl number, 281
Precedence rule, 17
Predictor-Corrector methods, 174;
178; 184; 195; 198
Prefixes, 2
Pressure, 1–3
Primary group, 289
Prototype, 282
Psychrometrics, 289–299; 308
See also: 447–452
Puffing, 289; 303–306
See also: 472–474

—Q—

Quadrature, 106; 116; 119; 122

—R—

Rank, 133
READ, 15; 19–28; 57; 67
Real Math Overflow, 198; 239
Real numbers, 14; 17; 62
Real Roots, 89; 102
Rectangular matrix, 136
Regression, 157–162; 164–166;
290; 304; 311
See also: 359–376
Regular geometry, 202; 204; 222
Relational operators, 23
Relative error, 238
Relative humidity, 290; 297; 309;
317
Relaxation, 149
Repetition, 13; 45
Residual value, 253
REWIND, 66; 74

Reynolds number, 281; 284
Richardson extrapolation, 82; 85;
109; 112
Robbins condition, 206; 210; 230;
304; 313
Romberg method, 106; 111; 115;
120
Roots, 89–102
See also: 329–340
Round-off error, 85–86, 108; 172;
239–240
Row vector, 126
Runge-Kutta method, 175; 184;
187; 191; 195; 198

—S—

Saturation vapor pressure, 290;
295
Scalar quantity, 239
Scaling up, 263; 282
Schmidt number, 281
Secant method, 90; 93; 98; 101
Second-order methods, 80; 84;
87; 108; 170–171; 174; 190;
199; 206; 219
Secondary group, 263
Shape functions, 247–248
Sherwood number, 241
Shooting method, 195
SI system, 1–3
Significant figures, 238
Similarity factor, 283
Simplex elements, 245–246
Simpson's rule, 106; 109–110;
113; 120–123
Slab, 202; 205; 208; 211; 222;
233; 290; 304; 313
Slug, 2; 8
Specific heat, 6; 291; 302; 304;
310
Specific volume, 290
Spherical, 202; 204; 208; 212;
216; 219; 222; 225; 290–291;
305; 311; 313; 318
Spurious solution, 101

Square matrix, 126; 130; 136; 151
 Stability, 204; 223; 229; 234; 302
 Standard error, 167
 Stanton number, 281
 STATUS, 36; 39–40
 Steady state, 212, 302–303
 STOP, 19
 Subprograms, 55–56
 SUBROUTINE, 55–56; 58; 66;
 69
 Subscripted variables, 32; 55; 62
 Subtraction (matrix), 126–127
 Subtraction, 16
 Surface area, 300; 310; 317
 Symmetry, 304; 313

—T—

Taylor Series, 78–79; 83–84; 108;
 110; 163; 172; 179; 239
 Temperature, 1–4; 263–264;
 266; 270
 Thermal conductivity, 8
 Thermal diffusivity, 203; 226
 Thin-layer, 290; 311; 316; 318
 Third-order method, 197
 Three-dimensional, 202; 235
 Time, 1; 263; 267; 277; 279
 Transcendental equation, 89; 125
 Transpose of a matrix, 130
 Trapezoidal, 106; 108; 115; 121
 Triangular element, 247; 250–251
 Triangular matrix, 127; 136; 141;
 142
 Tridiagonal matrix, 127; 151;
 222; 234
 Trigonometric regression, 162
 Truncation error, 108; 110; 114;
 172; 177; 181–182; 195; 239
 Two-dimensional, 228; 231; 234;
 247

—U—

Underflow, 239
 Units, 1

Unsteady state, 202
 Upper triangular matrix, 127;
 136; 141–142

—V—

Vapor pressure, 290; 294; 303
 Vector, 126; 135; 140; 144; 154
 Viscosity, 291; 295; 307

—W—

Wave equation, 202
 Weber number, 281
 Weights, 116; 120
 Wet bulb, 291–292
 Weighted residuals, 244; 253
 Work, 1–2
 WRITE, 15; 20–28; 38; 57

—Z—

Zeros, 89–90; 92